Titre: Title:	Proposition d'une méthode de développement d'ontologie pour un système expert en sécurité
Auteur: Author:	Simon Malenfant-Corriveau
Date:	2017
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Malenfant-Corriveau, S. (2017). Proposition d'une méthode de développement d'ontologie pour un système expert en sécurité [Mémoire de maîtrise, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/2922/

UNIVERSITÉ DE MONTRÉAL

PROPOSITION D'UNE MÉTHODE DE DÉVELOPPEMENT D'ONTOLOGIE POUR UN SYSTÈME EXPERT EN SÉCURITÉ

SIMON MALENFANT-CORRIVEAU
DÉPARTEMENT DE GÉNIE INFORMATIQUE ET GÉNIE LOGICIEL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

MÉMOIRE PRÉSENTÉ EN VUE DE L'OBTENTION
DU DIPLÔME DE MAÎTRISE ÈS SCIENCES APPLIQUÉES
(GÉNIE INFORMATIQUE)
DÉCEMBRE 2017

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Ce mémoire intitulé :

PROPOSITION D'UNE MÉTHODE DE DÉVELOPPEMENT D'ONTOLOGIE POUR UN SYSTÈME EXPERT EN SÉCURITÉ

présenté par : MALENFANT-CORRIVEAU Simon
en vue de l'obtention du diplôme de : Maîtrise ès sciences appliquées
a été dûment accepté par le jury d'examen constitué de :

M. DESMARAIS Michel C., Ph. D., président
M. FERNANDEZ José M., Ph. D., membre et directeur de recherche
M. GAGNON Michel, Ph. D., membre et codirecteur de recherche
M. GAHA Mohamed, Ph. D., membre

DÉDICACE

"C'est pourquoi, sème ton grain dès le matin et jusqu'au soir n'arrête pas de travailler. Tu ne sais pas quelle partie de ton travail réussira ou si tu tireras profit de toute ton activité."
(Ecclésiaste 11.6)

REMERCIEMENTS

J'aimerais tout d'abord remercier le créateur de l'univers de m'avoir donné la vie et de m'avoir permis d'étudier une petite partie très intéressante de sa création. Tout a été créé par lui et pour lui, et sans lui, rien n'est possible.

Je remercie ensuite particulièrement mon épouse, Marie-Ève, pour son soutien constant ainsi que ma fille, Élianne, qui a été un rayon de soleil dans ma vie durant toute la durée de mon travail. Mes remerciements vont aussi à Pierre-Luc Verville avec qui les multiples discussions m'ont intéressé à la recherche et m'ont aidé tout au long de la maîtrise, ainsi qu'à mes parents et beaux-parents qui m'ont soutenu par leurs encouragements et leurs prières.

Finalement, je tiens à remercier José Fernandez et Michel Gagnon pour leur direction précieuse ainsi que toutes les personnes du laboratoire SecSI, particulièrement Étienne Ducharme dont la compagnie et l'aide ont été très appréciées tout au long de mon passage à la maîtrise.

RÉSUMÉ

Depuis quelques années, l'idée de la représentation des connaissances par l'ontologie a émergée comme solution potentielle à la résolution de problèmes en sécurité informatique, particulièrement en ce qui concerne les systèmes experts de détection d'anomalies et d'intrusion. Entre autres, une proposition de son utilisation a été présentée dans le travail de Sadighian (2015). Notre travail a consisté à étendre le travail effectué dans cette thèse dans le but de concrétiser l'approche qui y est présentée, de manière à pouvoir faire une évaluation concrète de son applicabilité à l'industrie de la sécurité.

Au cours de cette entreprise, nous avons constaté que les ontologies présentées dans Sadighian (2015) étaient difficiles à appliquer dans un contexte d'implémentation de système expert. Pour cette raison, nous avons voulu développer de nouvelles ontologies plus concrètes à cette fin. Cependant, nous avons dû constater qu'il n'existe pas de méthode de développement d'ontologies ayant cette visée. Pour cette raison, nous avons cherché quelles devraient être les étapes d'une méthode permettant l'implémentation d'un système expert centré sur l'ontologie et avons développé ATOM ("Abstractions Translation Ontology Method"), une méthode en six étapes pour répondre à cette question :

1. Définition des requêtes de l'ontologie en langage naturel
2. Traduction des requêtes en SPARQL
3. Spécification des informations brutes
4. Création des étapes intermédiaires
5. Explicitation des règles de transformation
6. Enrichissement de l'ontologie

L'application de cette méthode permet de produire trois artefacts permettant l'implémentation d'un système expert :

1. Une ontologie
2. Un diagramme de traductions
3. Un document de spécifications

Nous avons par la suite énoncé cinq exigences sur lesquelles évaluer notre méthode :

1. Produire des artefacts permettant l'implémentation d'un système expert fonctionnel.
2. Pourvoir des guides et directions permettant de justifier les choix de modélisation et d'être plus efficace.

3. Donner une direction claire composée d'étapes séquentielles à effectuer.
4. Produire une ou des ontologies d'une qualité suffisante pour leur partage et leur réutilisation.
5. Être applicable dans une durée de temps raisonnable.

ATOM répond aux deux premières et à la quatrième exigences. Nos résultats sont cependant mitigés en ce qui concerne la troisième et la cinquième exigences.

Nous avons aussi voulu évaluer si la modélisation de l'information par l'ontologie profite d'un retour sur investissement en raison de la réutilisation des modélisations dans un domaine spécifique mais nous n'avons rien pu conclure à ce sujet. De plus, nous avons cherché à savoir si la méthode de représentation de l'information est assez flexible pour d'autres domaines de la sécurité et si on pouvait y retrouver ce retour d'investissement attendu. Pour ce faire, en plus du domaine de la détection d'intrusions en environnements informatiques de Sadighian (2015), nous avons appliqué notre méthode à la détection d'anomalies en contrôle aérien. Malheureusement, nous n'avons pas pu appliquer le processus au complet dans ce domaine et nous ne pouvons donc rien conclure au sujet de de nos questions inter-domaines.

Finalement, en plus de proposer une méthode de développement d'ontologies s'inscrivant dans le processus général de recherche au sujet de l'utilisation de l'ontologie en sécurité informatique, nous mettons en lumière l'utilisation de l'ontologie comme composante d'un système intelligent en sécurité en avertissant sur les dangers d'utiliser l'ontologie comme un simple graphe orienté, en expliquant de quelle manière il est possible d'utiliser des modèles d'apprentissage machine avec l'ontologie et en clarifiant comment utiliser le raisonnement sémantique permis par la représentation de l'information par l'ontologie.

ABSTRACT

From some time now, the idea of ontology based knowledge representation has emerged as a potential solution to resolve problems in computer security, particularly concerning anomaly and intrusion detection systems. Such a proposal was made in the work of Sadighian (2015). Our work had for initial goal to extend this proposal for it to be applied concretely in the industry of computer security.

While accomplishing this work, we noticed that the ontologies proposed by Sadighian (2015) were hard to apply in the context of an expert system implementation. For that reason, we decided to develop new ontologies for that specific use. However, we had to conclude that there is no existing method to do so. We then tried to find a sequence of steps for a method that could give the possibility of implementing an expert system and we obtained ATOM ("Abstractions Translation Ontology Method"), a six steps method to answer that need:

1. Natural language definition of requests
2. Requests translation in SPARQL
3. Raw data specification
4. Intermediary steps creation
5. Translation rules elicitation
6. Ontology improvements

The use of this method produces three artefacts that give the possibility to implement an expert system:

1. An ontology
2. A translation diagram
3. A specifications document

We then stated five requirements to evaluate our method:

1. It should produce artefacts that give the possibility of an expert system implementation.
2. It should give guidelines and rules of thumb that allow to justify models and to be more efficient.
3. It should give a clear direction composed of sequential steps to guide the process.
4. It should produce ontologies of sufficient quality to be shared and reused.
5. It should be applicable in a reasonable time.

ATOM answers the first, second and fourth requirements. Our results are however mitigated for the third and fifth requirements.

We also wanted to evaluate if the ontologies development process profits of a return of investment in a particular domain of security, for example the domain of intrusion detection in IT environments, but we could not conclude anything concerning this question. Also, we wanted to know if the proposed method was flexible enough to be used in other domains of security and if we could get a return of investment in multiple domains. To answer this question, we applied the proposed method to anomaly detection in air traffic control. Sadly, we could not apply the process completely and it is not possible for us to conclude anything concerning these questions.

Finally, in addition to the proposal of an ontology based expert system development method, we answered some questions concerning the place of the ontology in a system of artificial intelligence applied to security. While doing so, we warned against the use of the ontology as a mere oriented graph, we explained how we could use machine learning models jointly with ontologies and we clarified how to use the semantic aspect made possible by the use of ontology.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES ET ABRÉVIATIONS

IETF	Internet Engineering Task Force
ADS-B	Automatic Dependent Surveillance – Broadcast
ATOM	Abstractions Translation Ontology Method
CADAT	Cause, Action, Defense, Analysis and Target
CAPEC	Common Attack Pattern Enumeration and Classification
CEP	Complex Event Processing
CMS	Content Management System
CPE	Common Platform Enumeration
CVE	Common Vulnerabilities and Exposure
CVSS	Common Vulnerability Scoring System
CWE	Common Weakness Enumeration
DNS	Domain Name System
DoS	Denial-of-service attack
FTP	File Transfer Protocol
HTTP	Hypertext Transfer Protocol
IDMEF	Intrusion Detection Message Exchange Format
IDS	Intrusion Detection System
IP	Internet Protocol
IRI	Internationalized Resource Identifier
KDD	Knowledge Discovery and Data Mining
LD	Logique Descriptive
OVM	Ontology for Vulnerability Management
OWL	Web Ontology Language
PSM	Problem Solving Method
PSR	Primary Surveillance Radar
RDF	Resource Description Framework
RDFS	Resource Description Framework Schema
RDP	Remote Desktop Protocol
SDI	Système de Détection d'Intrusion
SPARQL	SPARQL Protocol and RDF Query Language
SQL	Structured Query Language
SSL	Secure Sockets Layer
STO	Situation Theory Ontology

SWRL	Semantic Web Rule Language
SWEBOK	Software Engineering Body of Knowledge
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UML	Unified Modeling Language
UPML	Unified Problem-solving Method description Language
VPN	Virtual Private Network
W3C	Worl Wide Web Consortium
XSS	Cross-site scripting

CHAPITRE 1 INTRODUCTION

Ces dernières années, le traitement automatique de l'information a pris un essor sans précédent. La plupart des échanges commerciaux sont maintenant numériques. On voit une augmentation de l'utilisation des systèmes experts, du "microtrading", des stations libre-service avec paiements intégrés dans les épiceries, les restaurants et les stations d'essence, des systèmes de proposition d'achats, des propositions de nouvelles à lire, etc. Bref, le logiciel est de plus en plus présent dans notre vie quotidienne. Il la façonne d'une multitude de manières souvent imperceptibles pour l'utilisateur final. Avec cette société de l'information viennent des problèmes auxquels l'humanité n'avait pas été précédemment exposée. En plus des multiples problèmes matériels et algorithmiques nouveaux par nature, d'anciens problèmes se présentent maintenant sous de nouvelles formes.

Dans ce travail, nous nous intéressons particulièrement aux problèmes de sécurité reliés aux systèmes d'information. Les problèmes relatifs à la sécurité ne datent pas d'hier, mais dans ce dernier siècle, ils se sont complexifiés. Dijkstra a écrit il y a quelques années que "By evoking the need for deep conceptual hierarchies, the automatic computer confronts us with a radically new intellectual challenge that has no precedent in our history." (Dijkstra et al., 1989) L'ordinateur en lui-même consiste en une intrication de multiples composantes et logiciels communiquant entre eux et impliquant l'interconnexion d'une grande quantité de travail qui exploite les connaissances de centaines, voire de milliers de personnes. Que dire maintenant des systèmes d'ordinateurs reliés entre eux qui présentent un enchevêtrement d'entités communicantes ayant chacune différentes caractéristiques et différents rôles sur leurs multiples réseaux ? On y retrouve plusieurs constructions logiques, fonctionnements, protocoles, langages, données, etc. La complexité du réseau informatique et la connaissance nécessaire pour une compréhension suffisante de sa structure dans son ensemble rendent ardu tout travail relatif à la sécurité.

Afin d'attaquer les problèmes de sécurité dans un monde informatisé complexe, on a développé une multitude de produits logiciels spécialisés dans l'analyse de l'information afin d'y détecter des dangers potentiels. Un de ces outils est le Système de Détection d'Intrusion (SDI), plus connu sous son nom anglais de Intrusion Detection System (IDS). La problématique initiale de laquelle découle le présent travail concerne les nombreux problèmes posés par cette catégorie d'outils (Ahmed, 2014), notamment le grand nombre de faux positifs qu'ils génèrent. Lorsqu'on fait l'analyse de ce problème spécifique, on conclut qu'il provient de la difficulté à prendre en compte la multitude d'informations des environnements dans lesquels ces outils

migrent. Plus particulièrement, ils peinent à bien représenter les connaissances et à les utiliser de manière efficace pour donner des résultats pertinents. En effet, lorsque les systèmes de détection d'intrusion détectent une attaque, ils ne savent pas reconnaître les attaques qui sont sans danger et génèrent de fausses alertes parce qu'ils n'ont pas de connaissance du contexte dans lequel les attaques sont exécutées (Sadighian, 2015). L'idée est donc d'apporter de l'information supplémentaire aux algorithmes de détection et c'est la représentation de cette information qui nous intéresse.

L'approche de représentation privilégiée dans le cadre de notre recherche est la représentation par ontologie, suivant le travail de Sadighian (2015). Bien que la représentation de l'information en utilisant l'ontologie ait connu des ratés dans le passé de la recherche en intelligence artificielle, cette idée a refait surface dernièrement, poussée par les efforts du World Wide Web Consortium (W3C). Cet organisme, sous l'égide de son fondateur, Tim Berners-Lee, fait la promotion du web du futur, le web sémantique (Berners-Lee, 2009). Cette proposition est accompagnée de choix technologiques et de formalismes que nous adopterons tout au long de ce travail. Le choix de l'ontologie comme formalisme de représentation de l'information se base sur certaines caractéristiques de cette technologie. Tout d'abord, la représentation par triplet de l'ontologie telle que présentée par le W3C a l'avantage d'être très simple à comprendre tout en permettant une expressivité satisfaisante. Ensuite, la possibilité d'intégrer du raisonnement logique en utilisant un vocabulaire approprié présente un atout par rapport à d'autres techniques comme le stockage dans des bases de données relationnelles. L'ontologie permet aussi de "faciliter l'échange de la connaissance et sa réutilisation pour toutes les parties intéressées par ce domaine particulier de connaissance" (Azni et al., 2008). De plus, avec la popularité internationale du "World Wide Web" et l'intégration croissante d'informations dans le web sémantique, nous croyons que cette technologie se démocratisera de plus en plus dans le futur. Finalement, l'utilisation de l'ontologie en sécurité informatique ne date pas d'hier et plusieurs études ont démontré ses avantages, comme on peut par exemple le lire dans Luh et al. (2017).

1.1 Problématique

La motivation initiale de ce projet était de concevoir des ontologies spécifiquement applicables dans le domaine de la sécurité informatique en les appliquant dans des environnements réels, de manière à étendre les résultats de Sadighian (2015). Au fil du processus, nous avons dû constater que "at present the construction of ontologies is very much an art rather than a science" (Jones et al., 1998). Cet énoncé revient souvent lorsque l'on étudie la question et nous l'avons nous-mêmes expérimenté dans notre travail. Lors de nos expériences de dévelop-

pement, nous avons aussi observé une différence entre les promesses de ces technologies et nos résultats. Par exemple, la promesse de réutilisabilité facilitée ne s'est pas concrétisée dans les résultats de nos développements. En effet, une ontologie partielle développée à un moment ne faisait plus de sens quelques mois plus tard et, malgré la documentation, il aurait été difficile de la réutiliser. Cette différence peut partiellement être expliquée par notre manque d'expérience initial. Cependant, elle est principalement la conséquence du suivi d'une méthode plutôt arbitraire qui nous faisait travailler beaucoup pour des raisons non justifiées. La raison de l'utilisation d'une telle méthode est qu'il n'existe pas de processus clair et concret que nous pouvions utiliser dans notre contexte. Ainsi, nous avons plutôt étudié le processus de développement appliqué à nos problèmes et nous avons développé une méthode qui donne des résultats concrets quant à l'implémentation d'outils utiles à l'industrie de la sécurité informatique.

1.2 Objectifs et questions de recherche

L'objectif principal de notre travail est donc de proposer une méthode de développement d'ontologies pertinente pour l'implémentation d'outils utilisés en sécurité informatique. Puisque l'ontologie n'est en fait qu'une manière de représenter l'information et que cette information n'a du sens que pour un "recognizer", pour utiliser un terme de Mark Stefik (Stefik, 1995), un objectif sous-jacent est de présenter l'architecture d'un système expert servant de cadre aux ontologies développées. Ainsi, nous avons choisi deux domaines d'application propices au développement d'outils en sécurité, soit la détection d'anomalies en contrôle du trafic aérien et la détection d'intrusions dans les systèmes informatiques. Puis, en développant les ontologies pour chaque domaine, nous avons fait la conception d'une méthode de développement d'ontologie de manière itérative, c'est-à-dire que nous avons ajusté la méthode au fur et à mesure que nous rencontrions des problèmes de modélisation ontologique.

Au fil du processus, nous avons tenté de répondre à cinq questions.

1. Quelles devraient être les étapes d'une méthode de développement d'ontologies utilisées dans le contexte d'un système expert en sécurité ?
2. Quelles sont les exigences qui assureraient à une telle méthode d'être efficace, reproductible et facile à apprendre et à mettre en oeuvre et est-ce que la solution proposée répond à ces exigences ?
3. Est-ce qu'un processus développé dans ce contexte serait assez flexible pour s'appliquer à plusieurs domaines de la sécurité ?
4. Est-ce que cette méthode bénéficierait d'un retour d'investissement dans la modéli-

sation de plusieurs types d'attaques dans un domaine en particulier ? Autrement dit, est-ce que plus on avance dans le processus de développement de l'ontologie, plus le temps de modélisation diminue en raison de la réutilisation des modélisations précédentes ?

5. Est-ce qu'elle bénéficierait d'un retour d'investissement semblable à la question précédente, mais cette fois, dans la modélisation de domaines différents ?

1.3 Structure du mémoire

Ce mémoire se divise de la manière suivante. Tout d'abord, dans le chapitre 2, nous présentons le paradigme scientifique de l'ontologie en expliquant en détail la signification portée par ce terme, puis en présentant les méthodes les plus populaires pour le développement de l'ontologie et son utilisation plus concrète dans le domaine de la sécurité informatique et du contrôle aérien. Par la suite, le chapitre 3 introduit l'architecture du système expert proposé qui sert de contexte pour l'ontologie, la méthode de développement proposée, ses principes fondateurs et un exemple qui l'illustre. Nous présentons ensuite dans le chapitre 4 nos résultats, c'est-à-dire les artefacts créés et les données recueillies lors du processus qui nous permettent de répondre à nos questions de recherche. Finalement, nous discutons de notre méthode et de nos résultats dans le chapitre 5, et concluons dans le chapitre 6.

CHAPITRE 2 L'ONTOLOGIE ET SON UTILISATION EN SÉCURITÉ INFORMATIQUE

"The term ontology is understood in a variety of ways and has been used in philosophy for many centuries. In contrast, the notion of ontology in the field of computer science is younger – but almost used as inconsistently, when it comes to the details of the definition." (Biemann, 2005)

L'ontologie est abordée de plusieurs manières par plusieurs domaines très variés, de la philosophie à l'informatique, en passant par la biologie. Les balbutiements de l'ontologie nous ont été présentés par la philosophie il y a de cela plusieurs centaines d'années, par exemple dans l'ouvrage "Les catégories" d'Aristote (Aristotle et al., 1960). Il a été depuis exploré par plusieurs auteurs, tels que Bertrand Russell (Russell, 1905), Rudolf Carnap (Carnap, 1997) et Quine (Quine, 1971). Le concept a été plus tard introduit en informatique en 1980 par un des instigateurs de l'intelligence artificielle, McCarthy, en référence à l'ontologie de Quine (Monnin, 2015). Plus récemment, le W3C poussé par son fondateur, Tim Berners-Lee, avance le concept de l'ontologie comme couche sous-jacente au web sémantique, l'extension proposée du "World Wide Web". On a aussi vu en 2001 la première proposition d'utiliser l'ontologie comme modèle de connaissance en sécurité informatique (Raskin et al., 2001).

On s'intéresse dans ce chapitre au paradigme du développement de l'ontologie. Nous exposerons tout d'abord sommairement ce qu'est la nature de l'ontologie, ses principes et ses attributs, ainsi que le formalisme proposé par le W3C dans le cadre du web sémantique. Par la suite, nous aborderons différentes procédures concernant son développement. Nous finirons le chapitre en présentant de quelle manière l'ontologie est utilisée dans la sécurité des environnements informatiques et en contrôle du trafic aérien.

2.1 L'ontologie

Selon Gruber, "une ontologie est une spécification explicite d'une conceptualisation" et "une conceptualisation est une vision du monde abstraite et simplifiée qu'on veut représenter pour un but donné" (Gruber, 1993). La représentation explicite de l'information occupe une place importante dans le développement logiciel. Pour citer Linus Torvald, "les mauvais programmeurs s'inquiètent du code" alors que "les bons programmeurs s'inquiètent des structures de données et de leurs relations." (Torvalds, 2006) Dans le cas du développement logiciel classique, on s'intéresse à des structures de données centrées sur les algorithmes, c'est-à-dire qui

permettront une facilité de compréhension et de maintenance ainsi qu'une efficacité du point de vue de la mémoire et du temps d'exécution. Ce qui intéresse le développeur de l'ontologie est plutôt une représentation centrée sur une conceptualisation répondant aux buts de l'ontologie. Cette conceptualisation se base sur le monde réel, car elle consiste en une vision simplifiée de celui-ci. Pour illustrer ceci, nous pouvons donner un exemple de Grady Booch sur des visions simplifiées du chat (Booch, 2006). Une propriétaire de chat ne s'intéressera qu'à certaines caractéristiques du chat, c'est-à-dire au fait qu'il ronronne, qu'il a besoin d'eau et d'un certain type de nourriture, etc. La vétérinaire aura une vision différente du chat, car elle s'intéressera à d'autres propriétés comme sa structure osseuse, ses organes internes, sa prédisposition à divers types de maladies, etc. On voit ici que dans les deux cas, les visions de la propriétaire et de la vétérinaire ne représentent pas parfaitement le chat, mais seulement les caractéristiques qui sont importantes pour chacune. C'est ce qu'on entend par conceptualisation. Dans ce cas-ci, une représentation plus complète du chat exposerait chacune de ses cellules, son phénotype, etc. La représentation complète est rarement nécessaire, voire impossible, et c'est pourquoi on s'intéresse seulement à une vision simplifiée du monde réel permettant de répondre à des questions spécifiques.

La représentation des connaissances par l'ontologie présente plusieurs avantages qui rendent cette technologie intéressante. Les attributs qu'on retient ici sont sa simplicité, sa flexibilité, la possibilité d'y appliquer du raisonnement et la possibilité de l'interroger à divers niveaux d'abstraction. Dans les sections qui suivent, nous présentons tout d'abord le formalisme proposé dans le contexte du web sémantique afin de démontrer sa simplicité. Ensuite, nous expliquons plus en détail ce qu'on entend par flexibilité. Puis, nous discutons de la logique descriptive qui permet de simuler le raisonnement dans l'ontologie. Nous finissons par démontrer l'avantage de l'abstraction rendue possible par ce type de représentation des connaissances.

2.1.1 Le formalisme du web sémantique

Comme mentionné plus haut, le W3C met en avant l'idée d'un web sémantique comme extension du "World Wide Web". Le cheval de bataille de cette proposition est un cadre permettant "d'exprimer de l'information au sujet de ressources" (Raimond et Schreiber, 2014) : RDF (Resource Description Framework). Ces ressources peuvent être ce que l'on veut, incluant des documents, personnes, objets physiques et concepts abstraits. Il propose aussi un langage de requête, SPARQL (SPARQL Protocol and RDF Query Language), permettant de récupérer les ressources des ontologies, de la même manière que SQL (Structured Query Language) permet de récupérer les données dans une base de données relationnelle. Nous

présentons les formalismes de RDF et SPARQL dans les sections qui suivent.

RDF

Le concept de base de RDF est très simple. Il permet de formuler des énoncés (“statements”) à propos de ressources sous la forme de triplets. Comme illustré dans la Figure 2.1, on appelle le premier terme le sujet, le deuxième terme le prédicat et le dernier terme l’objet.

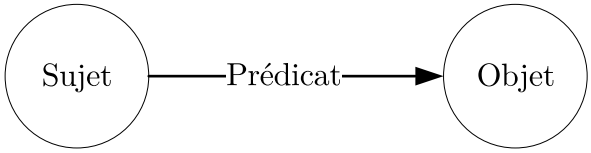


Figure 2.1 Triplet RDF

Ce formalisme est ce qu’on appelle la syntaxe abstraite de RDF. Il existe plusieurs syntaxes concrètes de RDF. Par exemple, en notation “turtle” on pourrait traduire la syntaxe abstraite du triplet de la Figure 2.1 par l’énoncé 2.1.

```
@prefix: <http://www.example.org#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
:subject :predicate :object.
:subject rdfs:label "Sujet"@fr .
:predicate rdfs:label "Prédicat"@fr .
:object rdfs:label "Objet"@fr.
```
(2.1)

Sans connaître en détail cette notation, on peut faire plusieurs remarques sur cet exemple. Tout d’abord, il y a deux types de ressources : des objets identifiés par des IRI (Internationalized Resource Identifier), tels que `:subject`, qui se traduit avec le préfixe par l’identificateur de ressources “http://www.example.org#subject”, et des littéraux tels que les chaînes de caractère “Sujet”, “Prédicat” et “Objet”. En réalité, le graphe représentant réellement la syntaxe concrète de l’énoncé 2.1 est présenté à la Figure 2.2. Dans la Figure 2.1, nous avons simplement remplacé les IRI des entités par leur étiquette (`rdfs:label`). On remarque de cette figure que RDF permet de représenter un graphe orienté. Un autre élément qui ressort de cette notation est l’idée d’espace de nom, où l’on utilise une abréviation comme `rdfs` pour définir une IRI de base qui englobe un vocabulaire. Finalement, on peut constater l’utili-

sation d'un vocabulaire prédéfini. Par exemple, on utilise le prédicat "label" de l'espace de nom RDFS (Resource Description Framework Schema). C'est cette idée de vocabulaire qui permet de passer de simples triplets RDF à une ontologie en bonne et due forme. En effet, RDF, RDFS et OWL (Web Ontology Language) sont des espaces de noms qui définissent un vocabulaire permettant de créer une ontologie.

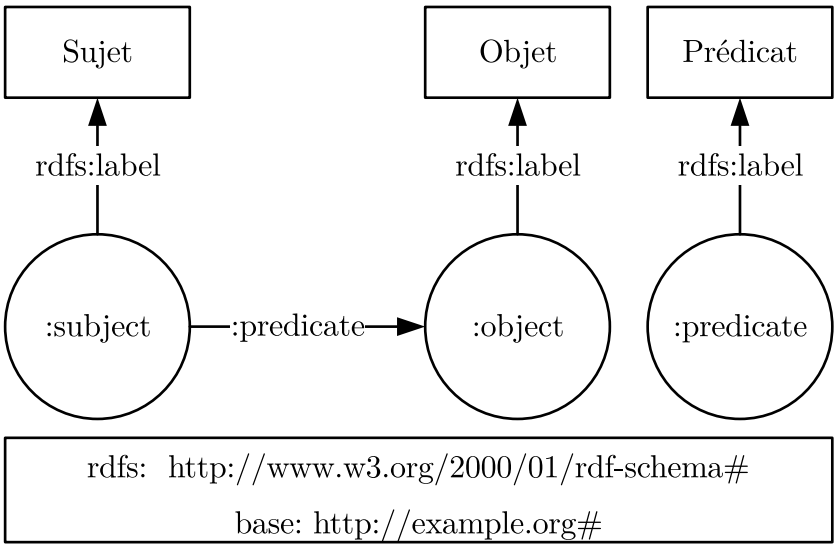


Figure 2.2 Exemple de RDF

C'est ici que se trouve le lien entre la définition de l'ontologie et RDF. Par exemple, RDF décrit la relation "type" qui permet d'instancier une classe. RDFS décrit des relations "subClassOf" et "subPropertyOf" qui permettent respectivement de créer des hiérarchies de classes et de propriétés. OWL décrit différents termes, comme "disjointWith", qui permettent de faire du raisonnement sur l'ontologie.

Pour illustrer de quelle manière est encodée l'information en utilisant le formalisme présenté, prenons l'exemple d'une taxonomie très simplifiée du monde informatique où l'on retrouve des ordinateurs, des serveurs et des services. On peut tout d'abord définir nos types d'objets. Pour ce faire, on utilise le concept "Class" de OWL pour définir les classes, "type" de RDF pour expliciter le type de classe d'un objet et "subClassOf" de RDFS pour définir les hiérarchies de classes. Nous écrivons cet exemple dans l'énoncé 2.2 en utilisant la syntaxe concrète "turtle".

```
:Machine rdf:type owl:Class .
:Serveur rdf:type owl:Class ;
         rdfs:subClassOf :Ordinateur .
:Service rdf:type owl:Class .
```
(2.2)

On peut ensuite définir la propriété “offre” de manière à pouvoir créer une définition de classe pour “Serveur”. En effet, un serveur est en fait toute machine qui offre au moins un service. On exprime cette définition en logique descriptive comme présenté dans l'énoncé 2.3.

$$Serveur \equiv Machine \sqcap \exists offre.Service \tag{2.3}$$

Il est aussi possible d'exprimer cette définition en utilisant le vocabulaire de OWL. Pour ce faire, on utilise le concept “ObjectProperty” pour définir la propriété et “intersectionOf”, “equivalentClass”, “Restriction”, “onProperty” et “someValuesFrom” pour définir une classe anonyme qui utilise le quantificateur existentiel. On démontre cette définition dans l'énoncé 2.4.

```
:offre rdf:type owl:ObjectProperty .
:Serveur rdf:type owl:Class ;
    owl:equivalentClass [
        owl:intersectionOf (
            :Machine
            [
                rdf:type owl:Restriction ;
                owl:onProperty :offre ;
                owl:someValuesFrom :Service
            ]
        ) ;
        rdf:type owl:Class
    ].
```
(2.4)

En utilisant cette formalisation, un raisonneur pourra par exemple inférer que tous les objets de type “Machine” qui “offrent” au moins un “Service” sont de type “Serveur”. De même, grâce à la hiérarchie de classe, pour tout objet déclaré comme un “Serveur”, le raisonneur inférera qu'il appartient à la classe “Machine”. Nous en dirons plus à ce sujet dans la sous-section 2.1.3.

SPARQL

Le langage proposé par le W3C qui permet l'interrogation d'un graphe RDF est SPARQL, acronyme pour "SPARQL Protocol and RDF Query Language". Ce que ce formalisme permet de faire est une correspondance ("matching") de motifs de sous-graphes avec le graphe de l'ontologie questionnée, où certains éléments de ce graphe de requête sont des éléments libres, c'est-à-dire des variables pouvant prendre une valeur du graphe dans lequel est effectuée la recherche.

Prenons par exemple le graphe RDF représentant une authentification VPN (Virtual Private Network), exprimé en "turtle" dans l'énoncé 2.5 et sous la forme de graphe à la Figure 2.3.

```
@prefix : <http://www.example.org#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
:VPNAuthentication rdfs:subclassOf :Event .
:sc3_vpnAuthentication rdf:type :VPNAuthentication .
:sc3_vpnAuthentication :hasDestinationIPAddress "10.5.1.200" .
:sc3_vpnAuthentication :hasSourceIPAddress "203.1.2.3" .
```
(2.5)

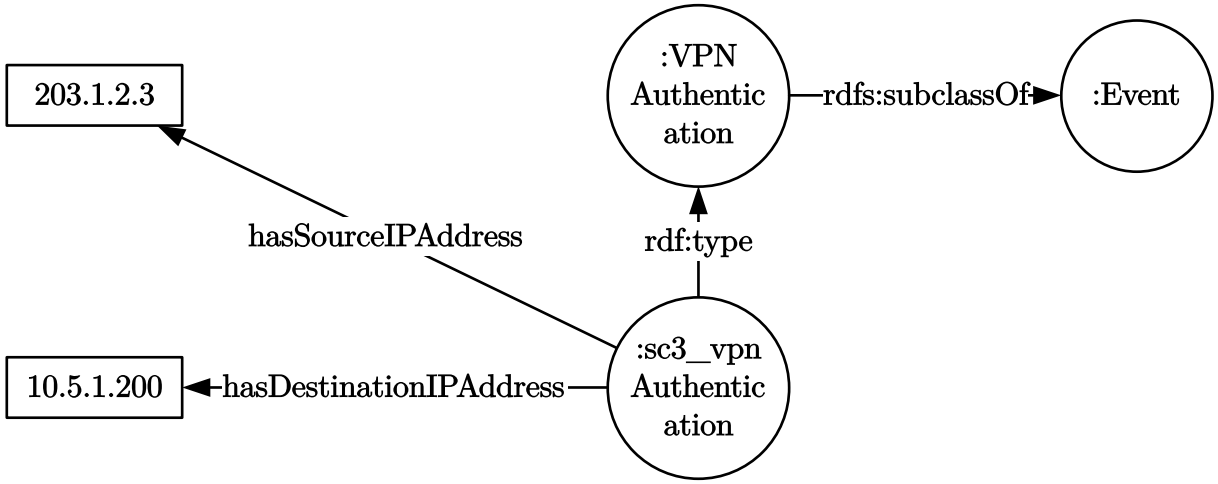


Figure 2.3 Exemple de graphe RDF

Il y aurait plusieurs manières d'interroger ce graphe, mais prenons tout d'abord le cas où nous voulons récupérer tous les événements de type "Event", en considérant qu'aucun raisonneur n'a inféré "`:sc3_vpnAuthentication rdf:type :Event`" en raison de la relation de sous-

classe entre “VPNAuthentication” et “Event”. La requête SPARQL pour ce cas spécifique est illustré dans l’énoncé 2.6.

```
SELECT ?event WHERE {
     ?event rdf:type ?directClass .
     ?directClass rdfs:subclassOf :Event .
}
```
(2.6)

Cette requête peut être représentée par le graphe de la Figure 2.4, où nous avons entouré les variables de lignes pointillées.

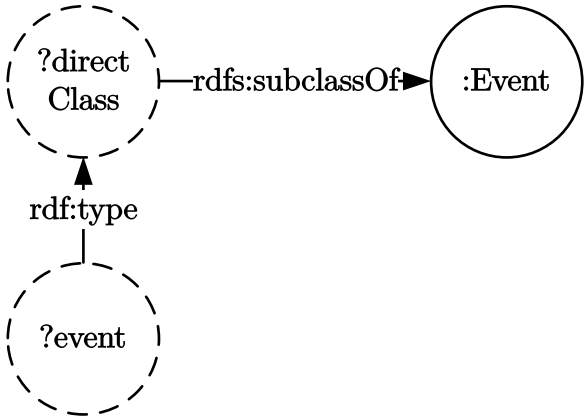


Figure 2.4 Exemple de requête SPARQL sous la forme d’un graphe

On constate ici que dans notre exemple, la variable `?event` peut être assortie à l’objet `:sc3_vpnAuthentication` et la variable `?directClass`, à l’objet `:VPNAuthentication`. Si nous avions voulu faire la requête directement sur la classe “Event”, comme présenté à la Figure 2.5, nous n’aurions obtenu aucun résultat puisque dans notre exemple, aucun élément n’est relié à la classe “Event” par la propriété “rdf:type”.

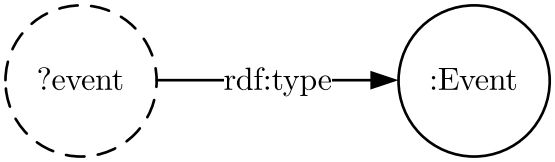


Figure 2.5 Exemple de requête SPARQL qui ne retourne aucun résultat

SPARQL permet de faire toutes sortes de requêtes utiles sur un graphe RDF pour y obtenir de l’information. Un autre exemple est le cas où l’on veut savoir si une requête d’authentification

VPN a été exécutée sur une adresse IP (Internet Protocol) spécifique, par exemple l'adresse 10.5.1.200. La requête SPARQL correspondante est écrite dans l'énoncé 2.7 et nous l'illustrons sous forme graphique à la Figure 2.6.

```
SELECT ?event WHERE {
     ?event rdf:type :VPNAuthentication .
     ?event :hasDestinationIPAddress ?destinationAddress .
     FILTER(?destinationAddress = "10.5.1.200") .
}
```
(2.7)

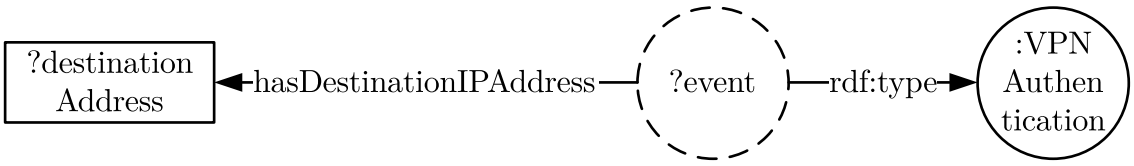


Figure 2.6 Requête SPARQL d'un événement d'authentification VPN effectué sur une adresse IP donnée

Dans ce cas-ci, la variable `?destinationAddress` prendra la valeur "10.5.1.200" et on introduit le terme "FILTER" de SPARQL qui permet de filtrer les résultats de la requête sur un critère donné. Ici, le filtre retournera une répondre positive et on obtiendra l'objet `sc3_vpnAuthentication` comme valeur possible de la variable `?event`.

Il est possible de spécifier des contraintes optionnelles dans la requête en utilisant le mot clé "OPTIONAL". Pour illustrer pour quelle raison cela est important, reprenons la requête 2.7. Supposons maintenant que nous voulions aussi récupérer un nom d'utilisateur associé à cet événement si cette information est spécifiée. La requête SPARQL modifiée est présentée dans l'énoncé 2.8.

```
SELECT ?event, ?username WHERE {
     ?event rdf:type :VPNAuthentication .
     ?event :hasDestinationIPAddress ?destinationAddress .
     ?event :hasUsername ?username .
     FILTER(?destinationAddress = "10.5.1.200") .
}
```
(2.8)

Si nous exécutons cette requête sur le graphe RDF de la Figure 2.3, nous n'obtiendrons aucun résultat, puisque l'événement `sc3_vpnAuthentication` ne possède pas la propriété "hasUsername". Pour retourner toutes les instances d'authentification, même si elles n'ont pas de nom d'utilisateur, et retourner le nom d'utilisateur lorsqu'il existe, on utilise le mot clé "OPTIONAL". La requête devient donc l'énoncé 2.9.

```
SELECT ?event, ?username WHERE {
     ?event rdf:type :VPNAuthentication .
     ?event :hasDestinationIPAddress ?destinationAddress .
     OPTIONAL {?event :hasUsername ?username . }
     FILTER(?destinationAddress = "10.5.1.200") .
}
```
(2.9)

Une dernière fonctionnalité que nous voulons souligner est la possibilité de regrouper les ensembles résultants de deux sous-graphes de requêtes. Par exemple, imaginons que nous souhaitions obtenir toutes les requêtes réseau utilisant le protocole TCP (Transmission Control Protocol) qui ont soit été effectuées sur le port 6667, soit sur le port 80. Il est possible de récupérer tous les résultats en une seule requête comme présenté dans l'énoncé 2.10.

```
SELECT ?event WHERE {
     ?event rdf:type :NetworkRequest .
     ?event :hasProtocol :TCP .
     { ?event :hasDestinationPort 6668 . }
     UNION
     { ?event :hasDestinationPort 80 . }
}
```
(2.10)

SPARQL pourvoit plusieurs autres fonctionnalités que nous n'expliquons pas ici en détail puisque notre utilisation se limite à ce que nous avons présenté. Le lecteur est invité à lire W3C (2013) qui les décrit plus en détails.

2.1.2 La flexibilité de l'ontologie

"The Semantic Web promises the kind of schema-flexible applications people need." (Karger, 2014) Un avantage de la représentation des connaissances par ontologie en utilisant le formalisme du web sémantique est sa grande flexibilité. Par flexibilité nous entendons la possibilité de réutiliser dans plusieurs contextes différents la même information. C'est-à-dire que, la plupart du temps, on peut réutiliser une ontologie développée en ajoutant un minimum d'information. Aucune modification n'est nécessaire. Prenons un exemple simple pour illustrer ce point.

Admettons une ontologie très simplifiée d'un réseau informatique et un peuplement de cette ontologie illustrant un réseau quelconque comme illustré à la Figure 2.7. Nous y définissons une hiérarchie de classes avec une classe de base "Machine" et une sous-classe "Serveur".

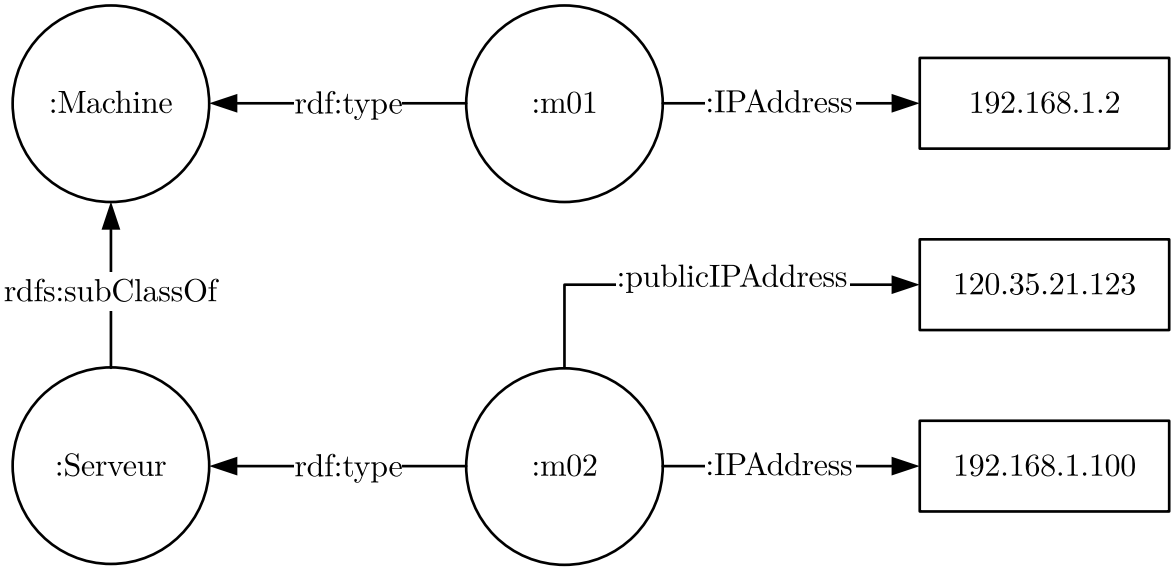


Figure 2.7 Exemple d'ontologie d'un réseau

On y retrouve ensuite deux machines, une ayant une adresse IP locale, et une autre étant un serveur et possédant une adresse locale et une adresse publique. Cette ontologie pourrait par exemple être la documentation d'un réseau et de sa configuration utilisable par un administrateur de système. Admettons maintenant un système de détection d'intrusion qui générerait ses alertes sous la forme d'une ontologie et qui aurait été exécuté sur le réseau décrit par l'ontologie de la Figure 2.7. Ce cas est illustré à la Figure 2.8.

On y décrit une hiérarchie de classe pour l'alerte avec comme sous-classe "Injection SQL". On y trouve aussi une instance d'alerte de type injection SQL qui a été détectée sur l'ordinateur ayant comme adresse IP 192.168.1.100. Ayant ces deux sources d'information, un

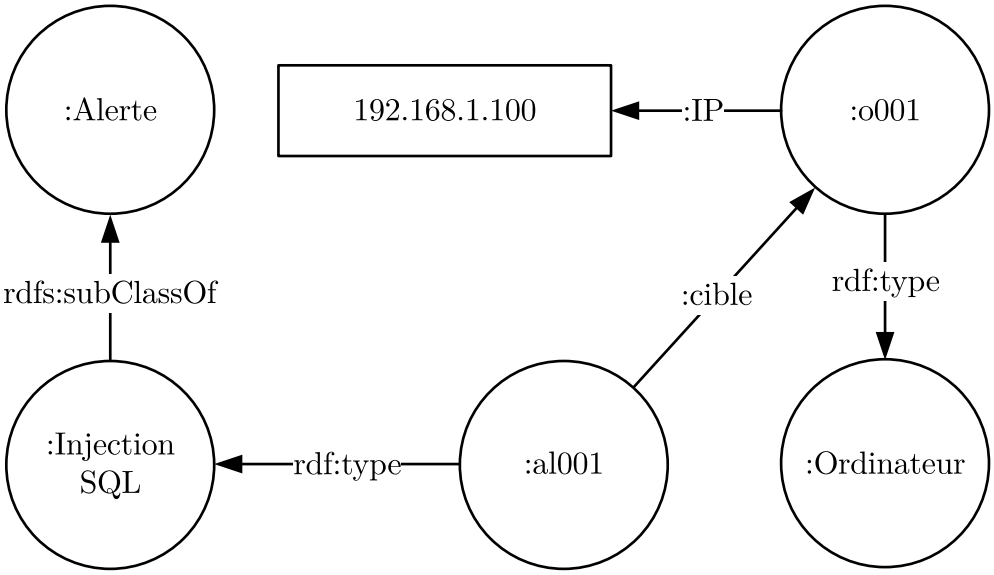


Figure 2.8 Exemple d'ontologie d'un SDI

administrateur pourrait vouloir plus d'information sur la machine ciblée par l'attaque détectée par le SDI. On voudrait alors pouvoir fusionner les deux ontologies dans la même base de données ontologique. En comparant les deux ontologies proposées, on voit qu'elles décrivent les mêmes choses, mais avec un vocabulaire différent. Pour regrouper les deux ontologies, il faudrait donc ajouter de l'information indiquant les termes synonymes. Il faudrait dire que :

1. Les propriétés `:IP` et `:IPAddress` sont synonymes
2. Les classes `:Ordinateur` et `:Machine` sont synonymes
3. Toutes les instances de type `:Machine` ayant une même adresse IP sont équivalentes

On obtiendrait alors l'ontologie partiellement illustrée dans la Figure 2.9.

On a regroupé les noms synonymes sous un même nœud, car, bien que cela ne soit pas vrai du point de vue de la base de données ontologique, ce l'est du point de vue de l'utilisateur par rapport aux requêtes qu'il peut exécuter. En interrogeant cette ontologie, l'administrateur de notre exemple pourrait par exemple constater que la machine attaquée est définie comme étant un serveur et il pourrait trouver son adresse publique en plus de l'information qu'il avait déjà dans l'ontologie du système de détection d'intrusion.

On peut conclure de cet exemple très simple que la combinaison de l'information représentée dans les ontologies est beaucoup plus facile et naturelle que si elle était représentée dans une base de données relationnelle, par exemple. Ce qu'on entend par "naturel" ici est que ce processus se rapproche beaucoup de la manière dont l'être humain emmagasine la connaissance.

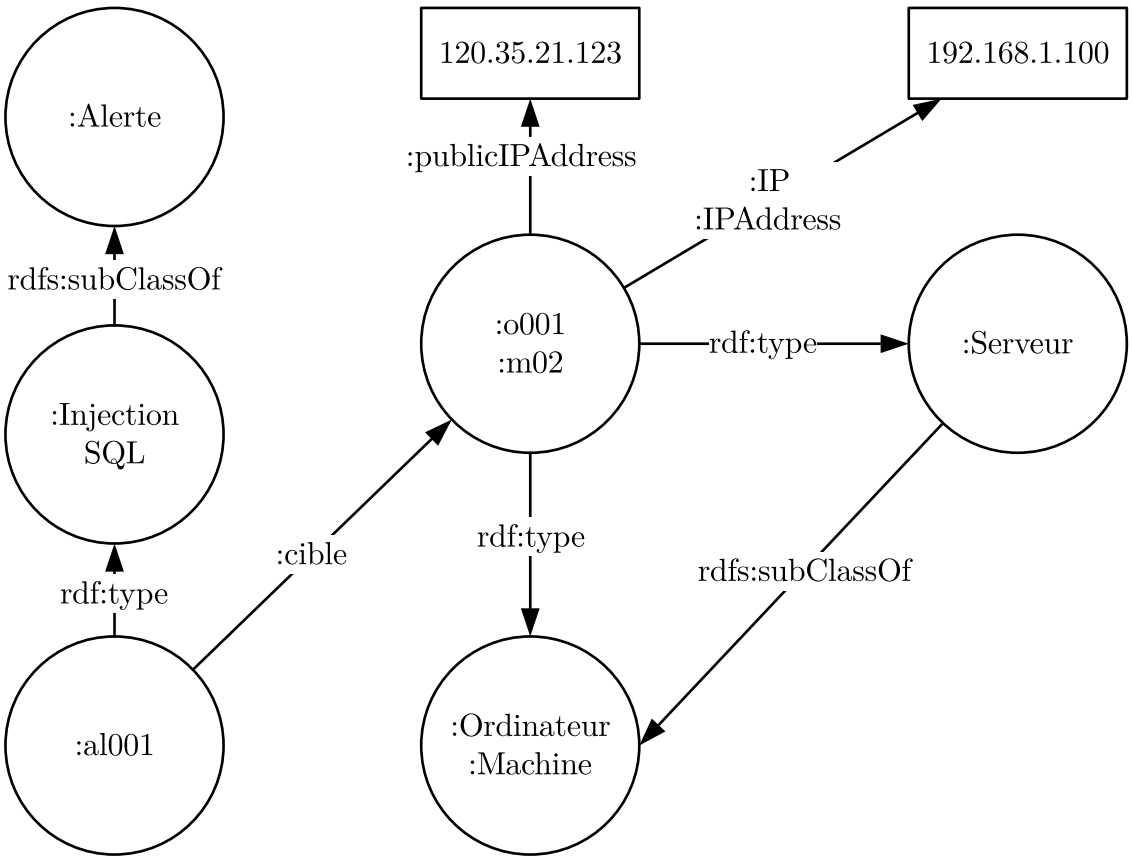


Figure 2.9 Combinaison des ontologies réseau et SDI

2.1.3 Simulation du raisonnement

On l'a vu précédemment, le paradigme de la représentation de l'information et de son traitement par l'ontologie est plus proche de celui de l'être humain quand on le compare aux autres manières de représenter l'information. De plus, la représentation par graphe orienté est simple et intuitive pour l'administrateur des données. Nous voulons maintenant démontrer de quelle manière on peut simuler le raisonnement fait de manière naturelle par l'être humain sur l'information qui la rend plus accessible et compréhensible. Comme le disent Stuart Russell et Peter Norvig :

"Les humains, semble-t-il, savent des choses, et ce qu'ils savent les aide à faire des choses. Ce ne sont pas des phrases creuses. Ils revendiquent à cor et à cri le fonctionnement de l'intelligence humaine, lequel découle non pas de processus purement réflexes, mais des mécanismes de raisonnement qui opèrent sur des représentations internes de connaissances." (Russell et Norvig, 2010)

C'est ce qu'on tente de simuler dans les ontologies en utilisant un vocabulaire prédéfini qui

permet de modéliser un système de logique descriptive. Nous avons abordé précédemment la notion de type, de sous-classe et de sous-propriété introduits par les vocabulaires RDF, RDFS et OWL. Ces vocabulaires voient leur plein potentiel d'utilisation seulement à la lumière des principes de la logique descriptive applicables à ceux-ci. L'idée est d'utiliser un algorithme permettant ce raisonnement logique, appelé raisonneur, qui pourra exécuter un processus d'inférence sur l'ontologie. L'inférence consiste simplement à générer de la nouvelle information à partir de l'information préexistante dans l'ontologie. Du point de vue de l'utilisation de l'ontologie en sécurité informatique, on peut considérer que le but de ces mécanismes d'inférence est double. Premièrement, elle permet des requêtes plus proches du langage des experts en sécurité ou des administrateurs de système qui utiliseront l'ontologie. Deuxièmement, elle permet la création de nouvelles informations qui n'auraient pas été nécessairement évidentes à n'importe quel utilisateur. Cela équivaut à créer un système expert où l'on essaie de simuler les conclusions qu'un expert en sécurité pourrait tirer des données, mais en utilisant seulement un système de règles logiques. Un exemple de cela dans le contexte des SDI est la génération d'alertes uniquement lorsqu'une attaque se produit dans un contexte permettant la réussite de l'attaque. Dans ce qui suit, nous présentons plus en détail la logique descriptive et les principes logiques de base utilisés pour faire des inférences dans l'ontologie.

La logique descriptive

La logique descriptive représente une famille de formalismes permettant la représentation d'un domaine de connaissance à l'aide d'énoncés logiques. Elle divise la connaissance en deux : l'ensemble des informations terminologiques, aussi appelée "T-Box", et l'ensemble des informations sur les individus, aussi appelée "A-Box". Cette première partie concerne toutes les "vérités" qui décrivent un domaine de connaissance. Par exemple, on pourra énoncer qu'un serveur est toute machine qui offre au moins un service. La deuxième partie, comme son nom l'indique, contient toutes les informations concernant des individus du monde modélisé. Par exemple, on pourra dire qu'une machine spécifique, la machine de Michel, est un serveur.

Les éléments de base de la logique descriptive sont le concept, le rôle et l'individu. Le concept permet de regrouper plusieurs éléments suivant des critères définis, c'est-à-dire qu'il décrit un ensemble. Le rôle, qu'on appelle aussi "propriété", sert à relier différents éléments entre eux sous la forme d'une relation un à un. À partir de ces éléments, on peut définir une sémantique qui se base sur la théorie des ensembles. En fait, celle-ci consiste à associer les entités du monde réel aux classes et aux rôles définis. Illustrons ceci par un exemple. Prenons le cas très simple où nous avons l'axiome 2.11 dans la "T-Box", qui indique que toute machine qui offre un service est un serveur.

$$Serveur \equiv Machine \sqcap \exists offre.Service \tag{2.11}$$

Admettons que nous avons un ensemble d'entités du monde réel $\{a, b, c\}$, où a et b sont des machines et où c est un service offert par a. On peut alors définir une relation d'interprétation I qui associe chaque entité à son ou ses concepts dans notre description. Dans notre cas, nous obtenons ce qui suit.

$$\begin{aligned} I(Machine) &= \{a, b\} \\ I(Service) &= \{c\} \\ I(offre) &= \{(a, c)\} \end{aligned} \tag{2.12}$$

Une interprétation valide de cette description est :

$$I(Serveur) = \{a\} \tag{2.13}$$

Une interprétation invalide selon le contenu de notre "T-Box" est :

$$I(Serveur) = \{b\} \tag{2.14}$$

De cette manière, nous arrivons à définir une certaine sémantique, autrement dit, un sens à des symboles, en dénotant les individus associés aux concepts de la description. Plus on affirmera de choses, plus on limitera le nombre d'interprétations possibles, et plus le "sens" des termes sera affiné.

On peut atteindre un niveau plus ou moins élevé d'expressivité en fonction des éléments qu'on utilise pour la description. Au plus bas niveau, qu'on caractérise par les lettres "$\mathcal{AL}$", il est possible d'exprimer les noms de concepts, les noms de rôle, la conjonction, et le quantificateur universel. Des niveaux supérieurs permettent d'exprimer la hiérarchie des rôles ($\mathcal{H}$), la négation de concepts ($\mathcal{C}$), la disjonction ($\mathcal{U}$), la quantification existentielle typée ($\mathcal{E}$), la restriction de cardinalité ($\mathcal{N}$), les rôles inverses ($\mathcal{I}$), la transitivité des rôles ($\mathcal{R}^+$) et d'autres extensions. Notons ici que les expressivités "$\mathcal{ALCR}^+$" et "$\mathcal{ALUER}^+$" sont notées "$\mathcal{S}$". Tous ces éléments ont pour but de spécifier plus en détail un domaine de connaissance et permettent à un raisonneur de faire des inférences sur les énoncés de la "A-Box". Si nous reprenons notre exemple, un raisonneur pourra conclure que a est un serveur puisque c'est le cas dans toutes les interprétations valides.

C'est pour permettre l'encodage des diverses expressions de la logique descriptive en RDF que le vocabulaire OWL a été introduit. Par exemple, on retrouve le terme "ObjectSomeValuesFrom" qui permet de créer des axiomes de quantification existentielle ($\exists$). OWL a été divisé en trois niveaux d'expressivités différentes : OWL-Lite, OWL-DL et OWL-Full. La raison à cela est que plus on augmente l'expressivité du cadre utilisé pour les modélisations en logique descriptive, plus cela rend l'inférence difficile. OWL-Lite correspond à la logique de description "$\mathcal{SHIF}$" et les algorithmes d'inférence pour ce niveau sont tous décidables. OWL-DL correspond à la logique de description "$\mathcal{SHOIN}(\mathcal{D})$". La différence avec OWL-Lite est que certaines résolutions seront faites en temps exponentiel. Finalement, OWL-Full implique la totalité des éléments de la logique descriptive et il consiste en une version indécidable de OWL.

Raisonnement sur les classes

Dans cette section, nous présentons quatre types de raisonnement de la logique descriptive s'appliquant aux classes.

Premièrement, on peut définir une classe par une clause d'équivalence à une classe anonyme définie par un énoncé logique comprenant des opérateurs logiques (et, ou) et des quantificateurs logiques. C'est ce qu'on appelle une condition nécessaire et suffisante de classification. Il existe deux types de quantificateurs : le quantificateur existentiel et le quantificateur universel. Dans le premier cas, on affirme que tous les objets en relation avec un autre objet d'un type donné par la relation spécifiée dans la quantification sont membre de la classe définie par celle-ci. C'est ce quantificateur qu'on a utilisé dans l'énoncé 2.11 lorsqu'on a défini un serveur comme étant la classe des objets de type "Machine" qui offrent au moins un service. Autrement dit, tous les objets de type "Machine" qui sont en relation avec au moins un objet de type "Service" par la relation "offre" sont de type "Serveur". Dans le deuxième cas, celui du quantificateur universel, on affirme que les objets appartenant à la classe définie par le quantificateur sont soit en relation par une propriété donnée avec des objets uniquement du type spécifié, soit n'ont aucune relation par cette propriété. Par exemple, on peut définir le carnivore comme étant la classe des objets qui mangent uniquement des animaux comme présenté dans l'énoncé 2.15.

$$Carnivore \equiv \forall mange.Animal \tag{2.15}$$

Autrement dit, une instance de carnivore ne peut être en relation par la propriété "mange" qu'avec des objets de type "Animal". Un carnivore pourrait cependant aussi être nourri par

intraveineuse et donc, il n'aurait aucune propriété "mange". Cela ne l'empêcherait pas d'être un carnivore, c'est-à-dire que dans ce cas-ci, tout ce qui ne mange rien serait classé comme un carnivore. Pour pallier cet inconvénient, on peut combiner plusieurs quantificateurs et classes en utilisant les opérateurs logiques. Par exemple, on pourra définir la classe des carnivores par des quantificateurs universel et existentiel pour obliger l'instance à avoir la relation "mange" comme présenté dans l'énoncé 2.16.

$$Carnivore \equiv \forall mange.Animal \sqcap \exists mange.Animal \tag{2.16}$$

Concrètement, l'utilisation de ces énoncés par un raisonneur va varier en fonction du contexte. Tout d'abord, prenons le cas où l'on crée un objet sans lui donner de type. Si on relie cet objet à un objet de type "Service" par la relation "offre", alors le raisonneur pourra inférer que cet objet est de type "Serveur" à cause de la quantification existentielle définie plus tôt. Si on lui ajoute une relation avec un objet de type "Animal" par la relation "mange", le raisonneur ne pourra pas conclure que l'objet est de type "Carnivore", si ce concept n'est défini que par la quantification universelle. Cela est dû au paradigme du monde ouvert utilisé par le raisonnement en logique descriptive. Étant donné ce paradigme, le raisonneur ne peut pas affirmer que l'objet n'est pas en relation avec quelque chose d'autre qu'un objet de type "Animal", parce que cela n'a pas été énoncé. Prenons maintenant le cas où l'on classifie manuellement les objets créés. Si on énonce qu'un objet est de type "Serveur" sans donner aucune autre propriété, le raisonneur pourra inférer que cet objet est en relation avec un objet de type "Service" quelconque par la relation "offre" sans qu'on le lui ait explicitement dit. Cela peut être utile dans les cas où un objet serait inféré par un autre mécanisme appartenant à la classe "Serveur" et qu'on voudrait récupérer tous les objets qui offrent un service. Pour ce qui est de la quantification universelle, il y a deux cas possibles. Tout d'abord, si on énonce qu'un objet de type "Carnivore" est en relation avec un objet de type "Végétal" par la relation "mange", le raisonneur conclura qu'il y a une inconsistance dans l'ontologie. Cela est utile lors du développement de l'ontologie pour détecter rapidement des problèmes de conception. Ensuite, le raisonneur peut inférer le type des objets impliqués dans la relation de la quantification universelle. Par exemple, si on affirme que l'instance "lion1" de type "Carnivore" "mange" l'objet "chose1", le raisonneur pourra inférer que "chose1" est de type "Animal".

Deuxièmement, on peut définir une classe par une condition nécessaire uniquement, c'est-à-dire que pour appartenir à la classe, un objet doit minimalement répondre à l'énoncé spécifié. Plus formellement, une définition de ce type affirme que la classe est sous-classe d'une classe anonyme définie par cet énoncé. Cela implique qu'il n'est pas possible de faire

de la classification automatique avec celui-ci. Prenons comme exemple le cas où l'on voudrait spécifier que tous les objets de type "Utilisateur" sont en relation avec des objets de type "Nom" par la propriété "aNom". Si on définissait cet axiome comme condition nécessaire et suffisante, cela aurait pour conséquence une classification automatique de tous les objets ayant un nom comme étant membres de la classe "Utilisateur". Cela ne serait pas un comportement acceptable, par exemple pour le cas d'une partition nommée. En définissant cet axiome comme condition nécessaire comme présenté dans l'énoncé 2.17, on spécifie seulement que les objets de type "Utilisateur" ont un nom et on prévient la classification automatique des instances qui en ont un. Pour ce genre de définition, on utilise les mêmes éléments axiomatiques que pour l'énoncé d'équivalence. De plus, les mêmes actions du raisonneur présentées pour le cas de l'équivalence s'appliquent pour ce qui est des instances classifiées à la main. On utilise donc cette construction lorsqu'on veut spécifier les relations qu'ont les objets d'une classe sans permettre au raisonneur de classifier une instance en se basant sur cette information.

$$Utilisateur \supseteq \exists aNom.Nom \tag{2.17}$$

Troisièmement, on peut utiliser des relations taxonomiques de sous-classification. Pour ce faire, on utilise la propriété "subClassOf" de RDFS. Par exemple, on peut spécifier que la classe "Serveur" est une sous-classe de la classe "Machine". On dit dans ce cas que "Machine" est une superclasse de la classe "Serveur". Ce que cette construction permet de faire concrètement est de l'inférence de type. Par exemple, si on crée une instance de type "Serveur", le raisonneur pourra inférer que cette instance est aussi de type "Machine". Ainsi, si on fait une requête de toutes les machines enregistrées dans le système, les objets qui ont été enregistrés comme serveur uniquement seront aussi dans les résultats. Ce cas de raisonnement est en fait similaire au deuxième cas mais avec une classe nommée au lieu d'une classe anonyme.

Quatrièmement, il est possible d'énoncer que deux classes sont disjointes. Cet énoncé implique qu'un même objet ne peut pas appartenir aux deux classes en même temps. L'utilisation concrète la plus évidente de cet axiome concerne la validation logique de l'ontologie. En effet, après avoir effectué son travail d'inférence, le raisonneur pourra vérifier une inconsistance si une instance se retrouve dans deux classes disjointes ou si deux classes disjointes sont inférées comme étant équivalentes.

Raisonnement sur les propriétés

On peut aussi définir les propriétés de l'ontologie d'une manière qui permet le raisonnement. Premièrement, comme pour les classes, il est possible de faire des hiérarchies de propriétés.

Cela est rendu possible par le prédicat “subPropertyOf” de RDFS. Par exemple, selon le contexte, on peut avoir des clés uniques variées. Un utilisateur aura son nom d'utilisateur comme identifiant alors qu'un ordinateur aura son nom d'hôte. Il est possible d'énoncer que les propriétés “aNomDUtilisateur” et “aNomDHote” sont sous-propriétés de “aIdentifiant”. Cela permet de généraliser les requêtes pour des types différents. Par exemple, si on récupère un ensemble d'ordinateurs et d'utilisateurs, on peut demander au raisonneur leur identifiant en utilisant seulement la propriété “aIdentifiant” même si cette information n'a pas été explicitée. Le système retournera les noms d'utilisateur pour les utilisateurs et les noms d'hôte pour les ordinateurs.

Deuxièmement, il est possible de spécifier des propriétés inverses. Par exemple, on peut énoncer que la propriété “aUtilisateur” a comme propriété inverse la propriété “aPourGroupe”. Pour démontrer l'utilisation de cet énoncé, prenons le cas où lorsqu'on instancie un utilisateur, on lui associe ses groupes avec la propriété “aPourGroupe” et que l'on ne crée pas cette relation lorsqu'on instancie les groupes. À cause de la relation inverse, on pourra demander au raisonneur de retourner tous les utilisateurs du groupe en utilisant la propriété inverse “aUtilisateur” même si elle n'a pas été énoncée. Les deux axiomes présentés jusqu'à présent permettent une meilleure navigation de l'ontologie et par le fait même, une plus grande facilité à modifier et à maintenir l'ontologie. Les deux éléments qui suivent permettent de faire de l'inférence de type.

Troisièmement, on peut utiliser le domaine (`rdfs:domain`) et la portée (`rdfs:range`) d'une propriété pour inférer le type des objets impliqués dans la relation. Le domaine permet d'inférer le type du sujet de l'énoncé. La portée permet d'inférer le type de l'objet de l'énoncé. Par exemple, on peut définir le domaine de la propriété “aSystèmeDExploitation” comme étant “Machine” et sa portée, “SystemeDExploitation”, comme illustré dans l'énoncé 2.18 en “turtle”.

```
:aSystemeDExploitation rdfs:domain :Machine ;
                       rdfs:range :SystemeDExploitation .
```
(2.18)

```
:chose1 :aSystemeDExploitation :chose2 .
```
(2.19)

Concrètement, cela implique que si on a un triplet 2.19, le raisonneur inférera que “chose1” est de type “Machine” et “chose2” est de type “SystemeDExploitation”. Ce type d'énoncé peut aussi permettre une meilleure compréhension de l'ontologie lorsque l'ambiguïté est possible. Par exemple, plus tôt nous avons introduit la propriété “aUtilisateur”. Notre but était d'avoir

une propriété qui permet de relier les groupes à leurs utilisateurs. Cependant, cet élément de la propriété n'est pas explicite dans son nom. En donnant un domaine de "Groupe" à cette propriété, on explicite ce que le créateur de la propriété avait en tête. Il faut cependant faire attention à ne pas confondre la restriction de type et l'inférence de type. En effet, si quelqu'un décide d'utiliser la propriété "aUtilisateur" pour, par exemple, relier un utilisateur qui s'est connecté sur un ordinateur à l'événement de connexion, cette personne aura la surprise que son événement sera aussi de type "Groupe". S'il est spécifié que les classes "Groupe" et "Événement" sont disjointes, l'ontologie sera considérée inconsistante. Si cela n'est pas spécifié, le raisonneur acceptera simplement qu'un groupe puisse être un événement. Donc, en résumé, on peut utiliser cette construction logique conjointement avec les axiomes de classes disjointes afin de valider l'ontologie, ou on peut l'utiliser pour inférer le type des objets impliqués dans une relation si cela n'a pas été explicité.

Quatrièmement, il est possible de spécifier un ensemble de caractéristiques aux propriétés. Il existe 4 types de propriétés, dont trois qui ont un inverse. Tout d'abord, il est possible de spécifier qu'une propriété est fonctionnelle. Cela implique qu'un sujet ne peut être relié qu'à un objet par cette propriété. Par exemple, la propriété "aTypeDOpération", qu'on utilise pour dire si un événement d'opération sur un fichier est en lecture ou en écriture, est fonctionnelle. En effet, un tel événement ne peut pas être en écriture et en lecture en même temps et il est inutile de spécifier plus d'une fois cette information. Cette caractéristique peut avoir concrètement deux implications pour le raisonneur. Premièrement, si un sujet est relié à plus d'un objet par une propriété fonctionnelle, le raisonneur conclura que tous les objets impliqués sont en fait le même objet. En effet, en logique descriptive, à moins qu'il soit affirmé explicitement que deux objets sont différents, ils peuvent être le même. Deuxièmement, si le raisonneur peut déterminer que les objets impliqués sont effectivement différents, il conclura à une inconsistance dans l'ontologie. On retrouve aussi l'inverse de cette caractéristique qu'on appelle "fonctionnelle inverse". Cette fois, c'est l'objet qui ne peut être en relation qu'avec un sujet par cette propriété. Par exemple, la propriété "aPageWeb" est fonctionnelle inverse, car un site web peut avoir plus d'une page, mais une page web ne peut pas appartenir à plus d'un site web en même temps. On peut noter ici que dans un autre contexte, il serait possible qu'une page web appartienne à plus d'un site web ou qu'un événement d'opération fichier ait le statut d'écriture et de lecture en même temps. On voit donc que les caractéristiques des propriétés peuvent permettre de spécifier formellement l'intention du créateur de la propriété, de la même manière que les domaine et portée des propriétés. Ensuite, on peut spécifier qu'une propriété est transitive. Par exemple, la propriété "aUnImpactSur" est transitive, c'est-à-dire que si un objet A a un impact sur un objet B qui a lui-même un impact sur un objet C, alors on peut conclure que l'objet A a un impact sur l'objet C. De cette manière, même si

la relation (A “aUnImpactSur” C) n’est pas explicitée, le raisonneur pourra affirmer que A a effectivement un impact sur C si on le lui demande. Le troisième type de caractéristique est la symétrie qui permet d’inférer un lien bidirectionnel entre des objets. Par exemple, la propriété “estMariéÀ” est symétrique, car si une personne est mariée à une autre personne, on s’attend à ce que cette dernière soit aussi mariée à la première. Il est aussi possible de spécifier qu’une propriété n’est pas symétrique. Par exemple, la propriété “aPartition” est asymétrique. En effet, bien qu’une machine puisse avoir une partition par la relation “aPartition”, une partition ne peut pas avoir pour partition une machine. Le quatrième et dernier type de caractéristique de propriété est la réflexivité. Cela implique qu’un objet relié à un autre objet par la propriété réflexive peut être relié à lui-même par cette même propriété. Par exemple, la propriété “estComposéDe” est réflexive, car un objet est habituellement composé de lui-même. On peut aussi utiliser la caractéristique inverse qui stipule qu’une propriété n’est pas réflexive. La plupart des propriétés entrent dans cette catégorie. Par exemple, un utilisateur ne peut pas être relié à lui-même par la propriété “aPourGroupe”.

La dernière affirmation qu’on peut faire au sujet d’une propriété est lorsqu’elle est une sous-propriété d’une chaîne de propriétés. Cela s’apparente à la caractéristique de transitivité, mais exploite une chaîne de propriétés différentes. Par exemple, la propriété “aOncle” est une sous-propriété de la chaîne “aParent o aFrere”. Cela implique que lorsque le raisonneur trouvera un objet A en relation avec un objet B par la relation “aParent” et que cet objet B est en relation avec l’objet C par la relation “aFrere”, il pourra inférer la relation (A “aOncle” C). Reformulé en français, on affirme que l’oncle de quelqu’un est le frère de son parent.

SWRL et les règles logiques

SWRL, acronyme pour “Semantic Web Rule Language”, permet d’exprimer des règles logiques sur une ontologie sous la forme d’une clause “antécédent → conséquence”, c’est-à-dire que si l’antécédent est exact, alors la conséquence doit aussi l’être. Par exemple, on peut affirmer que si un site web a un module qui a la vulnérabilité WPVDB-7846, alors ce site web a lui aussi la vulnérabilité WPVDB-7846. En utilisant la syntaxe de SWRL, on obtient l’énoncé 2.20.

$$\texttt{hasModule(?website, ?module)} \wedge \texttt{hasVulnerability(?module, WPVDB-7846)} \rightarrow \texttt{hasVulnerability(?website, WPVDB-7846)} \tag{2.20}$$

Comme pour SPARQL, les variables sont précédées d’un point d’interrogation. Ici, elles sont `?website` et `?module`. La conséquence de cette règle est qu’il sera ajouté à tous les

éléments `?website` répondant à l'antécédent de la règle la propriété "`hasVulnerability`" avec comme valeur "`WPVDB-7846`", de manière à ce que la conséquence soit elle aussi positive.

2.1.4 L'abstraction possible dans l'ontologie

Comme nous l'avons démontré dans la section 2.1.3, il est possible de créer de la nouvelle information à partir de l'information préexistante. Nous l'avons brièvement abordé précédemment, un des buts que nous recherchons dans l'utilisation de l'ontologie pour représenter la connaissance est de se rapprocher du langage courant de l'utilisateur de l'ontologie, soit dans le cadre de la sécurité informatique, celui de l'expert en sécurité ou de l'administrateur de système. Une caractéristique primordiale pour atteindre ce but est la capacité d'abstraction que présente l'ontologie. Nous en avons déjà glissé quelques mots, mais nous la présentons maintenant plus en détail en l'illustrant par des exemples.

Tout d'abord, prenons l'exemple d'une communication très simple entre deux individus dans le contexte de la menuiserie. Un contremaître pourrait simplement demander à un menuisier de poser une poutre à un endroit donné en pointant cet endroit. Dans ce seul ordre, il y a une très grande quantité d'informations implicites et abstraites. Attardons-nous simplement sur l'action de poser la poutre. Cette instruction est à très haut niveau et le menuisier doit la décomposer en actions plus concrètes s'il veut exécuter la tâche. Un niveau moins abstrait aurait pu être de prendre des clous, un marteau et la poutre, de la lever à l'endroit donné et de clouer les clous de telle ou telle manière dans la poutre. Un niveau encore moins abstrait aurait pu être de se déplacer jusqu'au marteau, de se pencher et d'utiliser son bras droit pour le prendre, de l'attacher à sa ceinture, de se relever, puis de se déplacer jusqu'à une poutre, etc. Un niveau encore moins abstrait aurait pu être d'utiliser tel ou tel muscle pour opérer le déplacement en utilisant telle quantité d'énergie, d'utiliser tel ou tel muscle pour ramasser le marteau, etc. On peut observer de cet exemple que plus on descend la hiérarchie d'abstraction, plus l'ordonnant doit communiquer longuement, c'est-à-dire plus il y a d'informations nécessaires à l'accomplissement du but. On observe aussi que le niveau maximal d'abstraction laisse une plus grande latitude au menuisier. Il doit inférer le but de l'ordre du contremaître et générer ainsi une série d'actions permettant d'atteindre ce but. On voit que ce mode de communication est très efficace, car il permet une forte délégation.

Ensuite, on peut prendre l'exemple du développement logiciel. Dijkstra a écrit que "by evoking the need for deep conceptual hierarchies, the automatic computer confronts us with a radically new intellectual challenge that has no precedent in our history." (Dijkstra et al., 1989) Il va même jusqu'à dire que "compared to that number of semantic levels, the average mathematical theory is almost flat" (Dijkstra et al., 1989) Lorsqu'on analyse la situation

actuelle de l'informatique, on peut conclure que les informaticiens ont bien relevé ce défi. Il est de plus en plus facile de développer des applications de plus en plus performantes à plusieurs niveaux, et cela à moindre coût. Une grande raison de ce succès provient des langages de haut-niveau et les librairies qui présentent des abstractions facilitant le travail des informaticiens. Nous pouvons par exemple citer le problème que représente les applications fenêtrées. Grâce à des librairies qui présentent les concepts abstraits de fenêtres, de positions et d'événements, le programmeur n'a souvent plus à gérer la carte graphique, la souris et le pixel à l'écran. Cette abstraction lui permet de se concentrer sur d'autres problèmes.

Finalement, nous pouvons illustrer l'avantage de l'abstraction du point de vue de la représentation des connaissances par l'ontologie. Dans ce cas, l'abstraction se traduit de deux manières. Premièrement, elle se manifeste par les propriétés "subClassOf" et "subPropertyOf", qui permettent de faire des hiérarchies de classes et de propriétés. Admettons la hiérarchie de classes présentée à la Figure 2.10.

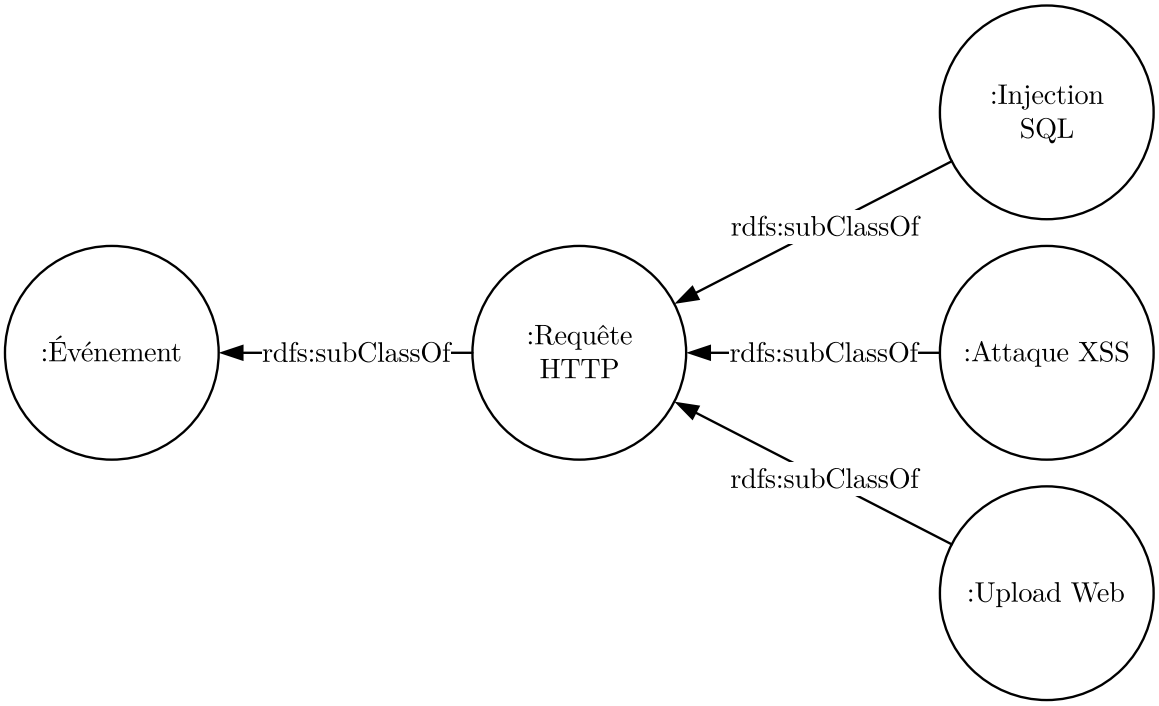


Figure 2.10 Hierarchie de classes

Cette hiérarchie présente trois niveaux d'abstraction allant de la gauche vers la droite, soit du plus abstrait au plus concret. Dans le contexte de la requête, plus on descend dans la hiérarchie, plus la réponse est spécifiée. C'est-à-dire que si l'algorithme fait une requête des instances de type "Injection SQL", il obtiendra uniquement les événements qui sont des requêtes HTTP (Hypertext Transfer Protocol) qui ont été détectées comme participant à

une tentative d'injection SQL. S'il demande plutôt les instances de type "Requête HTTP", il devrait obtenir toutes les requêtes HTTP du système, autant les injections SQL que les attaques XSS (Cross-site scripting) et les requêtes normales, etc. En fonction du besoin, la requête pourra donc être à des niveaux d'abstraction différents. Deuxièmement, dans le cadre des ontologies terminologiques, l'ontologiste peut définir son vocabulaire de manière à permettre l'abstraction. Par exemple, on s'attend à ce que le concept "Requête" soit plus abstrait que le concept "Requête HTTP".

2.2 Méthodes de développement de l'ontologie

La science du développement ontologique étant encore dans son stade premier, il existe plusieurs méthodes proposées dans la communauté scientifique. Celles-ci varient suivant le public cible et le contexte d'utilisation de l'ontologie. Nous en présentons trois des plus populaires ici et nous expliquons pourquoi elles s'appliquent difficilement à notre contexte.

2.2.1 Ontology Development 101

La méthode présentée par Noy et al. (2001) a pour public cible les nouvelles recrues du domaine de l'ontologie. Les auteures se sont principalement basées sur leur propre expérience de développement d'ontologies et ont emprunté des principes du monde de l'orienté-objet. Leur méthode est présentée avec l'utilisation de "Protege", un outil de modélisation d'ontologies. Bien qu'elle s'applique à une ancienne version de "Protege" qui traitait de la représentation par cadres, leur méthode est encore très utilisée. Elle est constituée de sept étapes.

La première étape consiste à déterminer la portée et le domaine de l'ontologie à développer. En effet, le vocabulaire sera fortement influencé par la portée de l'ontologie. Les auteures donnent l'exemple d'une ontologie dans le domaine des vins. Étant donné que le but de leur ontologie est de permettre un choix judicieux de vin en fonction d'un repas, elle ne considérera pas, par exemple, le prix du vin. Une activité particulière de cette étape est le développement de questions de compétence auxquelles l'ontologie devra pouvoir répondre. Ces questions font en quelque sorte office de spécification permettant l'évaluation de l'ontologie.

La deuxième étape consiste à réutiliser une ontologie existante autant que possible. Il est généralement admis que la réutilisation d'une structure de connaissance préexistante est moins coûteuse que la création ex nihilo.

La troisième étape consiste à énumérer les termes importants du domaine. Les auteures proposent comme technique d'expliquer le domaine à un néophyte. Les termes les plus importants ressortent ainsi inévitablement.

La quatrième étape consiste à définir les classes et leurs hiérarchies à partir des termes énumérés précédemment. Il existe trois méthodes pour ce faire :

— Top-down : partir des concepts les plus abstraits et créer les branches inférieures plus concrètes
— Bottom-up : partir d'instances plus concrètes et faire des regroupements pour aller progressivement vers les concepts plus abstraits
— Combinaison : trouver les concepts abstraits et concrets et relier les deux

La cinquième étape consiste à définir les propriétés des classes. Cette étape est dirigée par les questions de compétence puisqu'on modélisera uniquement les propriétés concordant avec la portée de l'ontologie. Il existe quelques types de propriétés :

— Intrinsèques : propriétés qui définissent directement un objet (ex. couleur)
— Extrinsèques : propriétés utilisées pour définir un objet, comme son nom ou sa provenance
— Parties : les différentes parties d'un objet s'il peut être structuré de cette manière
— Relations à d'autres individus : lorsque deux objets de différents types sont reliés, par exemple, le producteur d'un vin est relié au vin qu'il produit

La sixième étape consiste à définir les facettes des propriétés. Les facettes sont en fait les méta-informations des propriétés. Une propriété peut avoir plusieurs de ces informations :

— Cardinalité : indique le nombre de valeurs qu'une propriété peut avoir, une seule, plusieurs, au minimum un certain nombre ou au maximum un certain nombre.
— Type : une propriété peut être une simple chaîne de caractères, une valeur numérique, une valeur booléenne, une valeur énumérée ou une instance d'une classe.
— Domaine et portée : respectivement les classes de valeurs comme sujets et comme objets possibles pour une propriété. Par exemple, on peut dire que la propriété "manger" a pour domaine les êtres vivants et pour portée la nourriture. Il faut porter une attention particulière ici, car un système de raisonnement inférera que tous les objets de la propriété "manger" sont de la nourriture. Donc par exemple, un énoncé disant qu'un enfant a mangé du papier peut avoir la conséquence de faire entrer le papier dans la classe "nourriture".

Finalement, la septième étape consiste à créer les instances. C'est à cette étape qu'on crée les objets des classes définies plus tôt et qu'on fournit toutes les valeurs associées aux propriétés de ces classes.

2.2.2 Methontology

Cette méthode est couramment citée dans le domaine du développement ontologique. Elle a été proposée par Mariano Fernandez, Asucion Gomez-Perez et Natalia Juristo en 1997 (Fernández-López et al., 1997). Leur proposition découle de l'observation que le développement ontologique se rapproche souvent plus de l'art que du travail d'ingénierie. Pour améliorer la rigueur du processus de développement de l'ontologie, les auteurs affirment qu'il faut suivre une méthode standardisée. Pour ce faire, ils identifient une série d'activités exécutées lors du développement d'une ontologie. À partir de ces activités, ils proposent une méthode en sept étapes. Celles-ci sont conçues de manière à produire chacune un artefact. Les activités identifiées par les auteurs de la méthode sont les suivantes :

1. La planification des tâches principales, de leur durée et des ressources nécessaires pour les mener à bien
2. La détermination du but et de la portée de l'ontologie, c'est-à-dire du pourquoi de l'ontologie et de ses utilisateurs
3. Le listage des sources de la connaissance qui sera encodée dans l'ontologie et l'explication de comment cette information sera acquise
4. La conceptualisation d'un modèle du problème et de sa solution, c'est-à-dire la création de diagrammes abstraits permettant de passer de la définition à l'implémentation
5. La formalisation dans un système concret, par exemple en logique de premier ordre, en utilisant OWL, etc.
6. La recherche d'ontologies préexistantes permettant de simplifier le travail
7. L'implémentation de l'ontologie
8. L'évaluation de l'ontologie en la mettant à l'épreuve
9. La documentation de l'ontologie
10. La maintenance de l'ontologie

En se basant sur les dix activités énoncées, les auteurs proposent ensuite les étapes suivantes qui constituent leur méthode :

1. Spécification : une explication en langage naturel de l'ontologie comprenant le but de l'ontologie. Cette explication peut contenir des représentations intermédiaires, des questions de compétence, des contextes d'utilisation, des scénarios, des descriptions des utilisateurs et toute autre information utile pour décrire l'ontologie. L'artefact produit est le document de spécification des requis.

2. Acquisition des connaissances : consiste à récupérer l'information du domaine de l'ontologie en consultant des experts, en consultant des livres, des figures, des tables, en analysant des textes, en utilisant des outils d'acquisition de connaissances, etc. L'artefact produit par cette étape est le document d'acquisition de connaissance.
3. Conceptualisation : la construction d'un glossaire (concepts, instances, verbes, propriétés), de dictionnaires de données, d'attributs, de constantes, d'instances, de conditions, de règles et de formules. L'artefact produit par cette étape est le document du modèle conceptuel de l'ontologie.
4. Intégration : cette étape a pour but la réutilisation de concepts définis dans d'autres ontologies, l'utilisation de vocabulaire existant et la documentation de l'utilisation des concepts empruntés. L'artefact produit est le document d'intégration.
5. Implémentation : cette étape consiste à utiliser une technologie concrète quelconque telles que Prolog, un langage de programmation logique, ou Jena, une librairie Java de construction d'ontologies, pour rendre l'ontologie propre à l'utilisation, c'est-à-dire pour qu'elle puisse être peuplée et interrogée. L'artefact produit est le document d'implémentation.
6. Évaluation : vérification et validation de l'ontologie. Dans la vérification, on évalue si l'ontologie est correcte. Dans la validation, on évalue si l'ontologie répond à son but dans son cadre. Cette étape produit le document d'évaluation.
7. Documentation : la dernière étape consiste à produire des textes en langage informel expliquant l'ontologie développée, ses concepts, ses relations, ses utilisations potentielles, etc.

2.2.3 Méthode d'Uschold et Gruninger

Une autre méthode très citée est celle de Mike Uschold et Michael Gruninger (Uschold et Gruninger, 1996). Ils proposent une méthode en six étapes.

1. La capture des scénarios qui motivent la création de l'ontologie : ces scénarios consistent en des problèmes du monde réel sous la forme d'histoires ou d'exemples. Par la même occasion, il faut expliquer comment une ontologie pourrait régler ces problèmes.
2. Formulation de questions de compétence informelles : nécessite une stratification de questions du plus haut niveau au plus simple. En effet, une question générale en englobera souvent plusieurs plus spécifiques. Ces questions font office de spécification, car elles sont le lien avec les scénarios motivateurs.

3. Spécification de la terminologie en langage formel : par exemple en logique de premier ordre ou en logique descriptive
4. Formulation de questions de compétence formelles : cette étape consiste simplement à traduire les questions développées à la deuxième étape dans la spécification développée dans la troisième étape
5. Spécifications d'axiomes : définition des termes et des contraintes en langage formel permettant de répondre aux questions de compétence
6. Justification des axiomes et définitions à l'aide de la preuve de théorèmes de complétude

Un point particulier de cette méthode est que les auteurs proposent aussi trois lignes directrices lors du développement de l'ontologie.

— Clarté : documenter pour minimiser les ambiguïtés et pour bien se faire comprendre par les développeurs d'agents. Donner des exemples, définir formellement les concepts par axiomes, etc.
— Cohérence : utiliser un raisonneur pour s'assurer que les concepts internes soient logiquement exacts.
— Possibilité d'extension : la conception de l'ontologie devrait être faite en prévision des utilisations possibles dans le futur.

2.2.4 Critique des méthodes présentées

L'aspect positif des méthodes présentées est qu'elles dressent toutes un plan général des étapes à accomplir pour développer une ontologie. En particulier, elles présentent une première phase de spécification du problème et des buts recherchés par l'ontologie, suivi d'étapes qui permettent de passer de cette spécification à une ontologie implémentée. Cependant, elles sont trop générales pour nous aider concrètement dans le processus de développement. La raison à cela est à notre avis le grand nombre d'utilisations possibles de l'ontologie, de l'enseignement à son utilisation dans des systèmes automatisés. Par exemple, dans notre contexte, il est nécessaire de spécifier de quelle manière l'ontologie sera peuplée, étant donné que nous gérons plusieurs sources de données dynamiques. Malheureusement, seule la méthode Methontology présente cette étape comme faisant partie du processus.

À notre avis, certaines étapes proposées dans ces méthodes sont problématiques. La liste ci-dessous présente ces étapes.

— **L'énumération des termes importants :** Bien que cette étape puisse être utile jusqu'à un certain point, elle doit être plus élaborée que la simple génération d'une

liste de mots. Par exemple, on pourrait s'attendre à ce qu'une ontologie de la sécurité informatique ait un concept "Attaque". Cependant, cela n'est pas certain, car on pourrait décider de modéliser les attaques dans l'ontologie sous la forme de requêtes au système, rendant inutile la modélisation du concept en tant que tel.

— **La réutilisation d'ontologies existantes :** On l'a déjà écrit, l'ontologie peut être utilisée dans plusieurs contextes et, même pour un contexte et un domaine précis, il existe un très grand nombre de possibilités de modélisation. De plus, il n'existe pas de manière simple d'évaluer la qualité des ontologies existantes. Pour ces raisons, sans indication supplémentaire, cette étape nous semble difficile à mettre en oeuvre concrètement. Ces problèmes ne concernent cependant pas la réutilisation d'ontologies ou de vocabulaires plus larges qui sont très utilisés dans la communauté du web sémantique, comme par exemple Dublin Core. Ils concernent uniquement la réutilisation d'ontologies spécialisées. Par exemple, il ne nous a pas été possible de réutiliser la taxonomie AVOIDIT que nous présentons plus tard. En effet, bien que les concepts qui y soient présentés sont pertinents dans le domaine de la sécurité informatique, ils ne convenaient pas à notre utilisation de l'ontologie dans un contexte de système expert.

— **Définition des hiérarchies de classes et leurs propriétés :** Le problème principal avec cette étape est qu'aucune des méthodes proposées ne donne d'indice en ce qui concerne la justification des choix de modélisation. Cela semble être laissé à l'imagination des développeurs en ce qui concerne les utilisations potentielles de l'ontologie. Par exemple, lorsque vient le temps de modéliser concrètement la requête HTTP, les méthodes existantes sont de peu de secours pour nous permettre de décider s'il vaut mieux la modéliser sous la forme d'une instance de type "Requête" avec "HTTP" comme valeur de la propriété "aProtocole" ou comme une instance de type "RequêteHTTP". La raison à cela est qu'il n'existe pas de théorie de modélisation d'ontologies en fonction des besoins, contrairement à la modélisation des bases de données relationnelles, par exemple. Ce problème n'est pas abordé par les méthodes existantes.

— **Évaluation de l'ontologie :** En général, on ne conteste pas l'idée qu'il est nécessaire de tester tout modèle. Cependant, il n'est pas aussi simple de tester un modèle de données qu'un algorithme. Par exemple, on peut se questionner au sujet des tests d'une ontologie qui a pour but de faciliter l'enseignement de concepts d'un domaine spécifique. Les méthodes existantes offrent peu de secours à ce sujet.

Ensuite, on trouve quelques problèmes aux méthodes qui les rendent difficiles à utiliser. Nous les énumérons ci-dessous.

— **Inexistence de l'étape d'acquisition des connaissances :** Comme nous l'avons

expliqué plus haut, cette étape est inexistente dans la plupart des méthodes de développement d'ontologies alors qu'elle est nécessaire à notre contexte.

— **Découplage entre l'étape d'acquisition des connaissances et la formalisation de l'ontologie :** Lorsque l'étape d'acquisition des connaissances existe, elle est considérée à part de l'étape de modélisation alors qu'elle devrait avoir une influence majeure sur celle-ci. Par exemple, le fait que le système d'exploitation n'encode pas directement la signification de ses événements devrait influencer notre modélisation. Dans ce cas-ci, il faudra choisir si on modélise une classe "EvenementSystemeDexploitation" pour stocker directement les événements ou plutôt des classes plus abstraites comme "InsertionMediaExterne". Cette modélisation dépend fortement de la source d'information.

— **Flou sur l'utilisation du raisonnement :** Les méthodes ne permettent pas d'expliquer naturellement de quelle manière le raisonnement possible dans l'ontologie est utilisé. Cela a pour conséquence que plusieurs ontologies développées semblent plus être des graphes orientés avec une sémantique implicite associée aux termes plutôt qu'un système logique permettant à un raisonneur de répondre à des questions dans un monde ouvert.

— **Absence de formalisme pour les étapes de spécification :** Bien que toutes les méthodes soulignent l'importance de bien spécifier les requis de l'ontologie avant son développement, le formalisme à utiliser pour ce faire n'est pas clair et le lien entre cette spécification et les modélisations semble être laissé à l'improvisation des développeurs.

Bref, bien que les méthodes donnent une idée générale théorique du processus de développement d'ontologies, elles sont limitées lorsqu'on accomplit concrètement le travail de modélisation. Cela a pour conséquence l'improvisation des développeurs d'ontologies qui se basent sur leur logique, leur intuition et leur expérience. Il est possible que cette généralisation soit volontaire pour englober le plus de contextes possibles. Cependant, cela entraîne les difficultés de modélisation que nous avons évoquées et nous force à définir plus en détail un processus de développement appliqué à notre contexte.

2.3 L'ontologie et la sécurité des environnements informatiques

Au fur et à mesure du développement des technologies reliées aux ontologies en intelligence artificielle, la recherche en sécurité informatique s'est progressivement intéressée à ce domaine. En effet, plusieurs chercheurs font l'hypothèse que certains problèmes en sécurité informatique peuvent être résolus par une meilleure modélisation des contextes de ces problèmes. Comme mentionné plus tôt, la première proposition d'utiliser l'ontologie comme modèle de

représentation en sécurité informatique a été faite par Raskin et Nirenburg en 2001, où ils ont étudié les avantages de l'ontologie dans la représentation de leur domaine (Raskin et al., 2001). Depuis lors, l'ontologie a été étudiée dans plusieurs domaines de la sécurité informatique et a été utilisée pour implémenter plusieurs outils, comme des systèmes de détection d'intrusion, des antivirus et d'autres systèmes de détection de logiciels malveillants (Ahmed, 2014).

Comme nous l'avons mentionné plus tôt, les outils qui nous intéressent dans ce travail sont principalement les systèmes de détection d'intrusion. Notre travail a été instigué par celui de Sadighian et al., qui présente un système de corrélation d'événements permettant la détection d'intrusion (Sadighian, 2015). Les problèmes des outils existants qu'il identifie sont nombreux : la duplication d'alertes, le grand nombre de faux positifs, un grand nombre d'alertes non pertinentes, l'impossibilité de détecter des attaques de type "zero-day", l'impossibilité de détecter des attaques effectuées en plusieurs étapes et la nécessité d'une interaction constante avec l'être humain. L'idée derrière le système qu'il propose pour régler certaines de ces problématiques est d'utiliser l'ontologie pour représenter toutes les informations utiles à la détection d'intrusion. Ces informations sont encodées dans quatre ontologies : une pour modéliser les attaques, une pour les vulnérabilités, une pour l'environnement et une pour les événements. Cette séparation de l'information en ontologies suit une hypothèse de modélisation qui veut que les attaques génèrent des événements, que ceux-ci se produisent dans un contexte qui possède lui-même des vulnérabilités, ces dernières étant exploitées par les attaques. En reliant l'information de cette manière, les auteurs démontrent empiriquement qu'une exploration intelligente de l'ontologie permet de réduire le nombre de faux positifs.

Les auteurs de Elahi et al. (2009) proposent une ontologie en utilisant la syntaxe UML (Unified Modeling Language), qui permet lors des modélisations de système de prendre en considération les vulnérabilités ainsi que les attaques et les composantes affectées du système. Ils présentent ensuite l'utilisation de cette ontologie conjointement avec des cas d'utilisation UML, CORAS et i*, qui sont toutes des manières de modéliser des systèmes informatiques.

Dans Azni et al. (2008), les auteurs soulignent l'utilité de s'inspirer du système immunitaire pour faire la conception de systèmes informatiques, en expliquant qu'on peut y trouver des modèles pour la reconnaissance, la diversité, la mémoire, l'autorégulation, la protection dynamique et l'apprentissage. Ils affirment ensuite que l'ontologie est utile pour faciliter le partage et la réutilisation d'informations pour un domaine donné. À leur avis, l'ontologie présente principalement deux avantages pour la représentation de l'information relative à la sécurité des systèmes informatiques : une meilleure interopérabilité entre les divers systèmes de sécurité réseau et la possibilité de construire des ontologies plus spécialisées, c'est-à-dire

d'étendre facilement les modèles. Par la suite, ils expliquent que les systèmes immunitaires artificiels sont déjà utilisés pour des tâches comme la classification de documents, la robotique, la détection de la fraude, la reconnaissance de personnes et la détection d'intrusion automatique, d'où leur intérêt pour ces systèmes. Suit une section sur les motivations d'utiliser l'ontologie dans la modélisation des simulations de la sécurité des réseaux. Leur argument le plus important concerne la flexibilité, qu'ils appellent la "composabilité". C'est la capacité que donne l'ontologie de diviser un système en petites parties dont la somme compose son entièreté, petites parties définies au niveau sémantique. À ce sujet, ils affirment qu'il existe deux types de "composabilité" : syntaxique, en ce qui concerne les détails d'implémentation, et sémantique, en ce qui concerne la validité et l'utilité des systèmes composés. Des arguments sont aussi donnés en faveur du développement ontologique. Le premier avantage est que les techniques de gestion de la connaissance sont souvent inefficaces, car elles sont souvent enfouies dans le code et cela provoque des redondances. Ils affirment ensuite qu'un autre avantage est que le parcours de l'ontologie permet de découvrir des concepts et des relations entre eux, ce qui améliore la modélisation pour la conception de systèmes. La méthode de développement qu'ils ont utilisée se divise en six étapes : l'identification du but, la capture de l'ontologie, l'implémentation, le raffinement, les tests et la maintenance. Les questions à répondre pour l'identification du but sont : "Pourquoi l'ontologie est-elle construite ?", "À quelle fin sera-t-elle utilisée ?" et "Quel est le profil de ceux qui l'utiliseront ?". L'étape de la capture de l'ontologie se déroule en trois phases : la détermination de la portée de l'ontologie, la sélection d'une méthode de capture et la définition des concepts dans l'ontologie. Cela peut être atteint en posant des questions de compétence auxquelles l'ontologie est censée pouvoir répondre. Les auteurs affirment que le processus est semblable à celui de l'orienté-objet, mais qu'il se concentre sur les propriétés structurelles d'une classe plutôt que sur ses propriétés fonctionnelles. L'étape d'implémentation consiste à représenter de manière formelle l'ontologie en utilisant une syntaxe quelconque. Selon les auteurs, le développement d'ontologies est un processus itératif en quatre étapes : la définition des classes, l'arrangement des classes en taxonomie en utilisant soit une approche "top-bottom", "bottom-top" ou un mélange des deux, la définition des "facettes" et de leur valeur permise et la création d'instances. L'étape de raffinement est composée de deux phases : le raffinement "intracodage", qui consiste à corriger les erreurs à mesure que l'ontologie est implémentée et le raffinement "extracodage" qui consiste à faire de même, mais durant l'étape de test et l'utilisation réelle.

Les auteurs de Pereira et Santos (2012) justifient leur recherche en rappelant que les organisations comptent sur la performance de leurs systèmes d'information en ce qui concerne la plupart de leurs activités. Ces systèmes de plus en plus complexes génèrent de plus en plus d'informations relatives à la sécurité et ces informations sont souvent mal gérées. De plus, les

mécanismes reliés à ces informations sont souvent mal implémentés. Les auteurs proposent donc une ontologie du standard ISO/IEC 27001, un standard relatif à la sécurité de l'information, permettant son adoption par n'importe quelle organisation. Ils parlent ensuite d'un concept introduit par cette norme, les systèmes de gestion de l'information relatifs à la sécurité. Pour ce concept, la norme présente une méthode nommée PDCA qui permet d'établir, d'implémenter, d'opérer, de contrôler, de réviser de maintenir et d'améliorer la protection des informations nécessaires au fonctionnement des organisations. Ces atouts nécessitent une protection adéquate contre la perte de confidentialité, d'intégrité et de disponibilité en fonction d'une analyse de risque. Bien que ce travail soit relié à la sécurité de l'information, il ne l'est pas directement à notre champ d'intérêt, soit les systèmes de détection d'intrusion et d'anomalies. Cependant, cela démontre la grande variété de l'utilisation de l'ontologie en sécurité informatique.

Plus reliée à nos intérêts, on retrouve la thèse de Sherif Saad Mohamed Ahmed, qui présente un cadre pour améliorer les SDI en exploitant la représentation des connaissances sous la forme d'une ontologie OWL (Ahmed, 2014). L'auteur démontre un intérêt pour l'utilisation des langages de représentation des connaissances pour améliorer les SDI. Il se penche donc sur l'ontologie et sur les manières de l'utiliser pour les problématiques des SDI évoquées plus tôt.

Dans Fenz et Ekelhart (2009), les auteurs proposent une ontologie OWL pour décrire formellement le domaine de la sécurité. Ils divisent leur ontologie en trois parties : Sécurité, Entreprise et Location. Ils présentent ensuite les concepts et les relations clés de chaque ontologie et introduisent les axiomes en logique de premier ordre utilisés pour l'inférence. Ensuite, ils incorporent le "German IT Grundschutz Manual" dans leur ontologie, une base de connaissances du domaine de la sécurité de l'information. Ils ont par la suite évalué leur ontologie en suivant une version de la méthode de Uschold, M. et Gruninger, M. (1996) présenté à la section 2.2.3 dont les étapes sont :

1. Évaluation par un expert du domaine
2. Identification des questions de compétences informelles du domaine
3. Identification des questions de compétences formelles du domaine
4. Évaluation des questions sur l'ontologie

Dans Abdoli et Kahani (2009), une ontologie OWL est proposée pour améliorer les SDI actuels en ce qui concerne la détection d'attaques DoS (Denial-of-service attack). Leur architecture est basée sur plusieurs agents SDI qui écoutent les activités sur le réseau et un "MasterAgent" qui reçoit toutes les alertes des agents. Ils utilisent Jena et SPARQL pour implémenter le

modèle utilisé par les agents pour détecter les attaques. Ils ont testé leur approche sur KDD (Knowledge Discovery and Data Mining), un jeu de données utilisé pour tester les outils de détection d'intrusion. Ils ont obtenu de très bons résultats par rapport aux autres approches. Ils présentent cependant peu de détails sur l'utilisation concrète de l'ontologie dans leur système.

Pour régler les problèmes des systèmes de détection d'intrusion, les auteurs de Xu et al. (2009) proposent la collaboration et la corrélation. Ces deux techniques permettent de diviser la gestion de la sécurité des réseaux en trois étapes : la collection, l'évaluation et la corrélation des alertes. Ils affirment cependant qu'il y a un problème majeur concernant la collection des alertes. Ce problème est relatif à la représentation de l'information relative à la sécurité, à d'autre type d'information et à la connaissance de corrélation. Le but de leur travail est donc d'appliquer une ontologie de la sécurité pour uniformiser l'information permettant de faire la détection d'intrusion. Ils présentent l'ontologie comme une alternative à IDMEF (Intrusion Detection Message Exchange Format), un format de données utilisé pour échanger des informations relatives à la sécurité des environnements informatiques, puisque ce dernier permet la représentation uniformisée des alertes des SDI, mais pas de l'information externe. Les langages proposés sont RDF, RDFS, OWL. Ils proposent aussi l'utilisation de SWRL et RuleML, une syntaxe semblable à SWRL pour spécifier des règles logiques, de manière à étendre l'ontologie à l'aide d'information de comportement, c'est-à-dire d'axiomes et de contraintes. Leur méthode de développement de l'ontologie se divise en quatre étapes :

1. Modélisation au niveau conceptuel : concepts de haut niveau comme "Contexte", "Asset_Owner", "Vulnerability", "threat" et "countermeasure".
2. Description en OWL : ils utilisent Protege pour définir les hiérarchies de concepts
3. Définition de règles de corrélation avec SWRL : ils construisent des scénarios d'attaque
4. Définition de services de gestion de sécurité pour réponse automatique : Utilisation de OWL-S pour définir des processus d'intervention automatique

Dans Liu (2007) est proposé un système expert composé de connaissances de base en sécurité et de règles d'inférence pour évaluer les risques de sécurité. Ils affirment que l'implantation des technologies de défense (coupe-feu, SDI, etc.) n'est complète qu'avec l'établissement de politiques et de règles de sécurité, ce qui peut s'avérer ardu. Ils développent donc un modèle de connaissance pour faciliter ce travail, basé sur les standards de gestion de la sécurité. Comme plusieurs autres chercheurs, leur intérêt dans la représentation de l'information au moyen de l'ontologie se base sur son utilité pour décrire un domaine spécifique de connaissance et sur l'avantage de la réutilisation et du partage de cette connaissance. Ils présentent aussi Protege

comme outil de conception d'ontologies. Ensuite, ils présentent UPML (Unified Problem-solving Method description Language), une approche d'acquisition, de représentation et de partage de l'information basé sur l'ontologie. UPML est en fait une architecture pour décrire les systèmes basés sur la connaissance composée de trois parties :

1. Définition du problème qui devrait être réglé par le système basé sur la connaissance
2. PSM (Problem Solving Method), qui décrit le processus d'obtention de solution du système basé sur la connaissance
3. Modèle du domaine, qui décrit la connaissance à modéliser

Les avantages de cette architecture sont sa flexibilité et l'indépendance de ses composantes. En se basant sur cette architecture, les auteurs présentent trois ontologies principales pour la gestion des risques de sécurité de l'information :

1. Domaine : acquisition de la connaissance et modélisation des données et des failles de sécurité
2. Tâche : établissement des métriques et évaluation des risques
3. Résolution : utilisation d'heuristiques de résolution de problèmes

Dans Saad et Traore (2010), Sherif Saad, dont nous avons évoqué le travail plus tôt, présente, en collaboration avec Issa Traore, une ontologie pour représenter l'information d'analyse "forensic" d'intrusion de réseau et l'information de résolution de problèmes. Par "analyse forensic", nous entendons une analyse qui tente de détecter et de comprendre les intrusions une certaine période de temps après leur fait. Leur justification de l'utilisation d'une ontologie est que dans la vie réelle, les experts en sécurité se concentrent particulièrement sur la sémantique des activités malicieuses et non pas sur leur nature statistique, comme le font les approches récentes d'automatisation des détections d'intrusion. Ils soulignent aussi un problème des ontologies récemment proposées pour la sécurité "forensic" des réseaux : elles ont pour but de pourvoir un vocabulaire commun de manière à faciliter l'échange, mais sont moins utiles pour développer des systèmes experts ou de raisonnement. La raison à cela est que les ontologies ne pourvoient pas de mécanismes de résolution de problèmes : une ontologie de méthode qui permettrait d'accomplir une tâche en particulier. L'ontologie qu'ils proposent contient de la connaissance pour 11000 activités malicieuses et pour 30 méthodes de résolution de problèmes en "forensic" de réseau. Ils présentent les caractérisations d'ontologies "lightweight" et "heavyweight", basées sur les niveaux de formalité et de granularité de représentation de la connaissance. À leur avis, les systèmes intelligents tels que les systèmes experts ont besoin d'ontologies "heavyweight" et c'est ce qui est présenté dans leur recherche.

Pour développer leur ontologie, les auteurs ont utilisé l'approche Methontology de Lopez et Perez avec des éléments de l'approche de Uschold et King et de celle de Gruninger et Fox. Les étapes effectuées vont comme suit :

1. Spécification : élicitation des besoins auxquels l'ontologie répond, ainsi que ses caractéristiques ("heavyweight", type de logique, etc.)
2. Conceptualisation : identification des concepts et classes de base spécifiques au domaine. Ils ont identifié trois types de connaissance :
   (a) Les buts de résolution de problèmes : Les buts sont formulés sous la forme de questions de compétence informelles. Ils ont développé 71 questions de compétence, dont les suivantes :
      i. Quelles vulnérabilités existent dans le système ?
      ii. Quelles sont les ressources critiques ?
      iii. Suivant un ensemble de privilèges, qu'est-ce que l'attaquant peut faire ?
      iv. Quelles ressources sont vulnérables ?

      Ces questions sont distribuées dans une structure taxonomique où pour répondre aux questions parentes, il faut répondre à toutes les questions enfants. Ces questions servent de spécification à l'ontologie ("ontological commitment")
   (b) La connaissance de résolution de problème : ensemble des contraintes, rôles, et propriétés qui traitent la connaissance pour achever les buts de l'analyse "forensic"
   (c) La connaissance factuelle du domaine de la sécurité des réseaux : Classes, sous-classes, propriétés et relations entre les classes. Par exemple, "computer-virus", "webserver", etc.

   Pour créer une ontologie, il existe trois approches : "Top-Down", "Middle-Out" et "Bottom-Up". Ils ont utilisé l'approche "Middle-Out", qui consiste en l'identification des concepts les plus importants du domaine, suivi des concepts plus abstraits et plus spécifiques.
3. Formalisation et implémentation : Ils utilisent la logique descriptive, et en particulier OWL, pour formaliser leur ontologie.

Ils ont obtenu 111 classes, avec pour classes au plus haut niveau d'abstraction :

- "Attack"
- "Attacker"
- "Vulnerability"
- "Malicious"
- "Objective"
- "System"
- "Evidence"
- "Motive"
- "Privilege"
- "Impact"
- "Asset"
- "Location"

Ils présentent deux types de relations : les relations taxonomiques et les relations ontologiques. Les relations taxonomiques sont :

- "is-A"
- "subclass-Of"
- "superclass-Of"
- "instance-Of"

Les relations ontologiques sont :

- "Executes"
- "Has-A"
- "Gains"
- "Proved-By"
- "Traced-To"
- "Extracted-From"
- "Exploits"
- "Leaves"
- "Compromises"
- "Causes"
- "Has-A"
- "Uses"
- "Uses"
- "Requires"
- "Trigered-By"
- "Exist-In"
- "Located-At"
- "Target"
- "Elevates"
- "Affects"
- "Requires"

Après avoir présenté les concepts de leur ontologie, ils présentent le raisonnement sur l'ontologie. Ils affirment que le raisonnement est utile pour la généralisation, la prédiction, le diagnostic et pour tirer des conclusions à partir des faits. Trois types de raisonnement peuvent être implémentés sur une ontologie :

1. le raisonnement déductif : rendre spécifique une connaissance générale de l'ontologie. Ex. tous les A sont des B, x est A, donc x est aussi B
2. le raisonnement inductif : tirer des conclusions générales à partir des instances. Ex. n instances de type A, les mêmes n instances ont la propriété P, donc toutes les instances de type A ont la propriété P
3. le raisonnement abductif : basé sur un résultat, on peut inférer ce qui s'est passé avec les préconditions de ce résultat.

Le raisonnement déductif est le plus commun, mais les auteurs argumentent que les deux autres types peuvent être nécessaires pour certains raisonnements en "forensic". Ensuite, ils ont testé leur ontologie sur des attaques réelles perpétrées sur un "honeynet", un réseau possédant volontairement des vulnérabilités de manière à attirer les attaques, qu'ils ont rendu disponible en ligne. Ils ont identifié une attaque où un de leurs serveurs FTP (File Transfer Protocol) a été exploité pour exécuter une attaque DoS sur une machine externe au réseau. Ils expliquent ensuite comment utiliser leur ontologie pour reconstruire automatiquement cette attaque. Ils concluent en disant que les mérites de l'utilisation de l'ontologie pour leur sujet sont indéniables, mais que la construction et la maintenance de telles ontologies sont coûteuses en temps et efforts. Ils exploreront donc des approches pour faire cela de manière automatique dans le futur.

Les auteurs de Gao et al. (2013) utilisent l'ontologie pour permettre l'évaluation de la sécurité

des systèmes informatiques. Pour ce faire, ils classifient les attaques en cinq dimensions : impact, vecteur, cible, vulnérabilité et défense. Ils utilisent OWL et font des inférences avec des raisonneurs OWL et de Logique Descriptive (LD). Ils proposent une taxonomie pour les attaques classées selon les cinq dimensions énoncées plus haut et ils décrivent chacune de ces dimensions. Les composantes principales de l'ontologie sont :

1. Les classes qui représentent des concepts
2. Les relations qui représentent une association entre les concepts
3. Les fonctions, un cas spécial des relations
4. Les axiomes formels qui représentent des énoncés toujours vrais
5. Les instances qui représentent des individus dans l'ontologie

Ils affirment qu'il n'y a pas de méthode standard de développement, mais qu'il y a plusieurs "manières de faire" et règles. Ce disant, ils n'expliquent pas clairement comment leur ontologie a été développée.

Les auteurs de An Wang et al. (2010) ont développé une ontologie pour la classification des vulnérabilités qu'ils ont appelée OVM (Ontology for Vulnerability Management). Ils ont utilisé pour méthode de développement un mélange de méthodes pour le développement d'ingénierie des connaissances en logique descriptive et des conseils de développement ontologique de Gruber. Pour construire l'ontologie, ils ont utilisé CVE (Common Vulnerabilities and Exposure), CWE (Common Weakness Enumeration), CPE (Common Platform Enumeration) et CAPEC (Common Attack Pattern Enumeration and Classification), des sources d'information relatives à la sécurité informatique. Ils présentent ensuite leurs concepts au plus haut niveau d'abstraction, qui sont "Vulnerability", "IT_Product", "Attack", "Attacker", "Consequence" et "Countermeasure". Ils présentent ensuite un exemple de peuplement d'une vulnérabilité, d'un "IT_Product" et de quelle manière ils sont reliés. À leur avis, cette ontologie peut être utilisée de plusieurs manières. La première utilisation proposée est simplement un outil qui regarde les propriétés des vulnérabilités pour un logiciel donné et qui propose un logiciel similaire ayant moins de vulnérabilités. La deuxième utilise les scores CVSS (Common Vulnerability Scoring System) et d'autres propriétés des vulnérabilités pour un produit logiciel donné pour ordonner les attaques. La troisième consiste à évaluer la similarité entre les vulnérabilités. Pour ce faire, ils utilisent une hiérarchie de catégories pour les vulnérabilités, ce qui requiert l'utilisation de l'ontologie plus que les deux premières utilisations.

Les auteurs de Isaza et al. (2009) proposent un système expert basé sur deux ontologies et deux algorithmes d'apprentissage machine : "K-means" et un réseau neuronal. La première ontologie décrit les attaques, alors que la deuxième décrit les réactions que le système de-

vrait avoir lorsqu'une attaque est détectée. À l'écriture de l'article, ils avaient défini 1800 signatures d'attaque et 835 règles de prévention. Ils ont utilisé la méthode de développement "Methontology". Les ontologies ont été écrites avec OWL et SWRL. SPARQL a été utilisé pour mettre à jour les ontologies. Ils ont évalué leur approche de détection avec le "DARPA Data Sets Intrusion Detection Evaluation" ainsi que des attaques générées avec "Metasploit" et d'autres outils. Leur environnement de test n'est pas très bien défini, mais ils semblent obtenir de bons résultats de classification.

Les auteurs de Simmons et al. (2014) ont pour but d'améliorer CEP (Complex Event Processing), une méthode de traitement de séries d'événements qui utilise seulement la syntaxe, en introduisant une sémantique d'événements au moyen d'une ontologie. Cela permet de ne plus seulement vérifier des égalités syntaxiques entre les événements, mais aussi des égalités sémantiques. Ils abordent ainsi le problème de l'hétérogénéité des syntaxes d'une organisation à l'autre. Ils présentent trois cas d'utilisation où l'ontologie est utile. Le premier consiste simplement à regrouper les instances similaires sous une même classe, de manière à permettre la communication entre les systèmes hétérogènes. Cela est utile par exemple lorsqu'un produit possède des noms différents dans des systèmes qui doivent communiquer ensembles. Le deuxième cas permet de créer plus facilement les ensembles de corrélation, c'est-à-dire lorsque les instances ayant des attributs semblables sont dans des classes semblables ou dans les mêmes classes. Le troisième cas d'utilisation est de permettre de faire des liens entre les instances d'événements plus facilement. Par exemple, un produit peut nécessiter certaines fonctions machine pour être construit et chaque machine peut émettre ses fonctions disponibles. Ainsi, la relation entre le produit et la machine est basée sur la sémantique.

Les auteurs de Simmons et al. (2009) proposent la taxonomie "AVOIDIT", qui permet de classifier les attaques selon cinq catégories : "Attack Vector", "Operational Impact", "Defense", "Informational Impact" et "Target". Ils présentent plusieurs taxonomies précédemment créées ayant le même but, mais ils affirment qu'elles sont incomplètes et ne permettent pas un traitement efficace. Ils comparent effectivement les approches de classification avec le "SQL Slammer worm" et le "Microsoft RPC Stack Overflow", démontrant que leur méthode est plus exacte. Ils avancent la méthode "cause, action, defense, analysis and target", ou CADAT, pour faire la classification des attaques et présentent clairement comment utiliser cette méthode avec l'ontologie proposée. Les étapes proposées sont :

1. Classifier le vecteur d'attaque
2. Classifier l'impact opérationnel
3. Classifier les défenses potentielles
4. Classifier l'impact informationnel

5. Classifier la cible de l'attaque

Ces étapes correspondent aux classes de leur taxonomie, c'est-à-dire qu'ils ont respectivement une classe représentant les vecteurs d'attaque, une représentant les impacts opérationnels, etc. Ils soulignent qu'ils ont respecté les requis suivants, de manière à ce que leur taxonomie soit universellement acceptée (basé sur un travail de Howard et al.) :

1. Acceptée : construite sur des travaux précédents.
2. Mutuellement exclusive : chaque attaque peut n'être classifiée que dans une seule catégorie
3. Compréhensible : claire et concise.
4. Complète/exhaustive : chaque catégorie est complète, c'est-à-dire que toutes les attaques peuvent être classifiées.
5. Non ambiguë : les classes sont clairement définies.
6. Répétable : la classification d'une attaque d'une fois à l'autre donne toujours les mêmes résultats.
7. Termes bien définis : la terminologie doit être bien définie et compatible avec la communauté en sécurité.
8. Utile : permet d'apporter quelque chose de nouveau ou une meilleure compréhension du domaine.

Ces points sont importants, car peu de papiers du domaine de l'utilisation de l'ontologie en sécurité informatique présentent des critères d'évaluation de la qualité des résultats. Les auteurs présentent leur taxonomie dans le contexte d'un système de résolution de problèmes qu'ils ont partiellement implémenté.

Les auteurs de van Heerden et al. (2012) proposent une ontologie permettant la classification de scénarios d'attaque de réseau. Ils présentent tout d'abord une taxonomie composée de :

— "Actor" — "Actor Location" — "Aggressor"
— "Attack Goal" — "Attack Mechanism" — "Automation Level"
— "Effects" — "Motivation" — "Phase"
— "Scope" — "Target" — "Vulnerability"

Ils proposent ensuite une ontologie basée sur cette taxonomie. Les concepts les plus importants autour desquels l'ontologie est construite sont :

— "Denial Of Service" — "Industrial Espionage" — "Password Harvesting"
— "Spear Phishing" — "Web Deface" — "Snooping for secrets"
— "Finacial theft" — "Cyber warfare" — "Industrial Sabotage"
— "Amassing computer ressources"

Ils décrivent ensuite en détail les relations entre les classes en utilisant les termes de leur taxonomie. L'utilisation concrète de cette ontologie n'est pas explicitée.

Les auteurs de Simmons et al. (2014) proposent un système de détection et de résolution de problème relié à la sécurité informatique qui se base sur une ontologie. Leurs concepts de haut niveau sont la taxonomie "AVOIDIT", les risques, l'organisation, la complexité et les ressources. Ils ont implémenté leur système et l'ont testé avec des experts en sécurité de diverses organisations. Un accent particulier a été porté sur la transmission de la connaissance relative aux événements problématiques.

Dans Simmonds et al. (2004), les auteurs présentent une ontologie très légère des concepts de sécurité suivants :

— "Access" — "Actor" — "Attack" — "Impact"
— "Information" — "Intangible" — "Motive" — "Outcome"
— "Systems Administrator"

Les relations introduites sont :

— "assesses" — "causes loss of" — "gains" — "has"
— "loses" — "makes" — "reports" — "uses"

L'ontologie contient un vocabulaire pertinent à la description des attaques de systèmes informatiques. Les concepts de l'ontologie sont un peu décrits, mais il n'est cependant pas expliqué comment l'ontologie est peuplée ou utilisée.

Les auteurs de Busch et Wirsing (2015) proposent aussi une ontologie légère, mais cette fois, ils documentent spécifiquement les concepts de sécurité relatifs aux applications web. Leur ontologie a pour but d'enseigner et d'aider les développeurs lors de la conception et de l'implémentation des applications web. L'ontologie est principalement basée sur le "SecEval's Security Context model". Pour leur ontologie, ils instancient les concepts "Secrity Property", "Methods", "Vulnerabilities" et "Threat". Ils donnent un exemple d'utilisation avec les relations du "cross-site-scripting" (XSS). Ils ont utilisé l'ontologie dans le contexte d'un tutoriel ayant pour but d'enseigner la sécurité des applications web et dans le contexte d'un standard d'ingénierie, "UML-based Web Engineering", qui permet de documenter de manière visuelle les décisions relatives à la sécurité lors du développement des applications web. L'accent de ce travail est spécifiquement sur le transfert de l'information entre êtres humains plutôt qu'entre machines ou systèmes informatiques.

Les auteurs de Martimiano et Moreira (2005) présentent une ontologie légère ayant pour but de représenter les termes et les relations reliés au domaine des incidents de sécurité. Ils présentent quelques questions de compétence : "Quels virus peuvent utiliser quelles vulnérabilités ?", "Quels sont les incidents reliés ?", "Quelles sont les conséquences d'un incident

spécifique ?" et "Quelles sont les ressources impactées par un incident ?" Ils affirment avoir utilisé la méthode Methontology, puisqu'elle est la plus mature, étant basée sur IEE Standard 1074-1995, un standard de développement logiciel. Leurs concepts sont :

— "Access" — "Agent" — "Asset" — "Attack"
— "Consequence" — "Vulnerability" — "Time" — "Tool"
— "Security Incident"

Leurs propriétés sont :

— "acts_on" — "happens_on" — "implies_to_a" — "proceeds"
— "precedes"

Leur ontologie a été évaluée par un étudiant qui a développé un "plug-in" OWL pour interfacer Protege avec Snort, de manière à peupler l'ontologie avec les alertes de Snort.

Les auteurs de Arbanas et Čubrilo (2015) répertorient 52 ontologies créées dans le contexte de la sécurité de l'information. Ils ont classé ces ontologies en trois catégories : les ontologies générales, les ontologies spécifiques à un sous-domaine de la sécurité et les travaux théoriques. Comme sous-domaines, on retrouve :

— les incidents de sécurité
— l'analyse de l'impact sur les entreprises
— la sécurité des réseaux
— le facteur humain
— la sécurité physique
— la sécurité des applications web
— les attaques de services web
— l'éducation de la sécurité de l'information
— les métriques de la sécurité de l'information
— le développement logiciel
— les attaques
— l'infonuagique
— les applications mobiles
— les requis de sécurité
— les vulnérabilités
— les standards en sécurité

Les ontologies présentées ont plusieurs portées, de l'éducation au système expert. Encore une fois, cela démontre la grande variété de l'intérêt pour l'ontologie dans le domaine de la sécurité.

On peut conclure cette section sur l'utilisation de l'ontologie dans la sécurité des environnements informatiques en affirmant qu'il existe plusieurs manières d'utiliser les ontologies dans ce domaine et plusieurs méthodes pour les développer. Certains chercheurs expliquent plus en détail que d'autres comment ils utilisent et développent les ontologies qu'ils présentent. Cependant, on doit constater qu'aucun résultat de recherche rapporté dans cette section ne nous permet d'engager une équipe de programmeurs pour implémenter un système expert. En effet, les étapes sont souvent trop vaguement définies ou trop génériques pour que nous puissions les utiliser concrètement. De plus, les méthodes de développement présentées nous

apportent peu de secours pour régler les problèmes de modélisation concrets rencontrés au fil du processus. Finalement, on ne retrouve aucune justification des concepts créés et leur utilisation est souvent laissée à l'imagination du lecteur. Ce que cela signifie est que, par exemple, il est assez évident qu'une ontologie de la sécurité informatique devrait avoir un concept "Attaque". Mais aucun chercheur n'explicite comment les instances de cette classe sont créées et comment elles sont utilisées dans les réponses aux problématiques abordées. Par le manque de justification, nous signifions que les choix des concepts semblent arbitraires. Par exemple, les chercheurs ne présentent pas pourquoi ils ont modélisé une instance de type "Requête" ayant "HTTP" comme valeur de la propriété "aProtocole" au lieu d'une instance de type "RequêteHTTP". Nous soulignons cependant le travail de Ahmed (2014) dont le travail a une portée concrète de système de détection d'intrusions et où l'utilisation de l'ontologie est plus explicitée que dans plusieurs autres travaux. Cependant, il n'est pas clair de quelle manière la méthode Methontology a pu être utilisée concrètement dans le contexte de l'étude.

2.4 L'ontologie et la sécurité du contrôle du trafic aérien

Un autre domaine où l'utilisation de l'ontologie à des fins de résolution de problèmes de sécurité est prometteuse est celui du contrôle du trafic aérien. Bien que cet intérêt soit présentement plutôt marginal, nous croyons qu'il aurait avantage à être développé plus en profondeur et nous présentons ici quelques efforts de recherche qui ont été accomplis dans cette direction.

Les auteurs de Fenza et al. (2010) s'attaquent à la problématique de "situation awareness", c'est-à-dire de la compréhension de l'état d'un environnement, afin de faciliter le processus de prise de décision dans le domaine de la sécurité des aéroports. Ils présentent la modélisation par ontologie comme une solution à ce problème, étant donné qu'elle permet de modéliser un ensemble d'objets en relation les uns avec les autres dans un environnement dynamique. Pour construire leur ontologie, les chercheurs ont utilisé comme base l'ontologie STO ("Situation Theory Ontology"), qu'ils ont étendue pour l'adapter au contexte spécifique de la sécurité des aéroports. Ils utilisent ensuite une architecture basée sur des agents pour effectuer un raisonnement sémantique sur l'ontologie. À leur avis, leur solution présente plusieurs avantages. Par exemple, ils affirment que leur solution de modélisation est simple et s'étend facilement à d'autres situations, qu'elle améliore les processus décisionnels relatifs à la sécurité des aéroports et qu'elle facilite l'échange de données entre les aéroports.

Dans Meyer et al. (2013), les auteurs combinent CEP et une ontologie pour permettre la gestion des systèmes de contrôle du trafic aérien. Ils affirment qu'il y a deux raisons pour

lesquelles il n'est pas possible d'utiliser uniquement l'ontologie pour accomplir cette tâche. Tout d'abord, la résolution d'un système écrit en logique descriptive par un raisonneur est NP-complet, ce qui implique une utilisation de trop de ressources pour que ce soit viable. Ensuite, à leur avis, il n'y a pas de manière de modéliser efficacement des séries de temps. C'est pour cette raison qu'ils combinent la modélisation par ontologie avec CEP, qui s'occupe de l'agrégation des événements. Leur proposition a été testée dans le contexte d'une reconfiguration automatique d'un radar. Ils ont trouvé que la combinaison de l'ontologie avec CEP a permis de faciliter l'analyse du système pour deux raisons. La première est que la modélisation sémantique de l'ontologie permet de décrire un système hétérogène de manière homogène. Ce qu'ils entendent par cette affirmation est que l'ontologie leur permet de modéliser des informations avec des caractéristiques très diversifiées (source, granularité, niveau d'abstraction, etc.) dans un seul système assez souple pour en faire une gestion uniforme. La deuxième est que CEP permet de gérer dynamiquement un grand flux de données.

Bien que leur travail ne soit pas relié à l'automatisation de la gestion de l'information, les auteurs de Su et al. (2011) démontrent que l'ontologie peut être très utile pour la modélisation des connaissances relatives au contrôle du trafic aérien. En effet, le but de leur travail est de faire la conception d'un système d'apprentissage qui vise à former les employés du contrôle du trafic aérien dans le but d'améliorer leur travail et de diminuer les erreurs. La modélisation de la connaissance sous la forme d'une ontologie leur permet de gérer les ressources d'apprentissage de manière plus efficace. Leur système a été évalué qualitativement par des utilisateurs et les chercheurs considèrent que leur but a été atteint.

Le problème qui est abordé dans Moser et al. (2009) est le défi de pourvoir une plate-forme flexible de services de gestion dans le contexte de la gestion du trafic aérien. Selon les auteurs, la difficulté est principalement d'intégrer un grand nombre d'informations hétérogènes qui ne sont pas toujours disponibles dans un format utilisable par la machine. Ils proposent donc une approche basée sur la connaissance utilisant l'ontologie comme modèle de stockage pour régler ce problème. Le but de leur ontologie est de modéliser la connaissance des experts du domaine afin de permettre l'intégration de l'information provenant de divers systèmes. Elle permet donc de relier cette connaissance, qui est spécifique aux systèmes impliqués, à la connaissance générale du contrôle du trafic aérien. Une étude de cas industrielle a permis aux auteurs d'obtenir deux résultats probants. Premièrement, leur modélisation de l'information relative au contrôle du trafic aérien permet d'automatiser certaines tâches relatives à l'intégration de systèmes en contrôle du trafic aérien, particulièrement en ce qui concerne les étapes de vérification. Cela a entre autres pour impact une diminution des erreurs potentielles d'intégration. Deuxièmement, ils démontrent que leur approche réduit le temps d'intégration et de maintenance des systèmes de gestion du contrôle du trafic aérien.

Dans Klauza et al. (2014), des chercheurs font référence à ADS-B (Automatic Dependent Surveillance – Broadcast), une technologie de surveillance qui permet de récupérer diverses informations de géolocalisation des avions en vol. Étant donné que ce système se base sur des antennes à bas prix, il est particulièrement propice à la contribution participative, mais les auteurs soulignent qu'il n'existe pas de manière standard de partager cette information. Ils proposent donc d'utiliser les technologies du web sémantique pour unifier les données provenant des récepteurs ADS-B. Selon les auteurs, l'utilisation de la modélisation par triplets RDF et par l'ontologie permettrait de combiner l'information provenant de sources multiples de manière efficace. De plus, cela faciliterait la recherche d'information par l'utilisation de SPARQL. Cependant, pour tester entièrement leur solution, il faudrait qu'ils développent l'ontologie, travail qui leur reste à faire.

On peut conclure de cette courte revue de littérature de l'utilisation de l'ontologie en contrôle du trafic aérien que l'idée est pertinente, puisque d'autres chercheurs s'orientent dans cette direction. Entre autre, Su et al. (2011) démontrent que ce domaine est applicable à la représentation de l'information par l'ontologie. Nous notons aussi que les études présentées explicitent de quelle manière l'ontologie est utilisée dans la résolution de leurs problématiques. Cependant, de la même manière que pour l'utilisation de l'ontologie en sécurité informatique, les auteurs n'expliquent pas clairement comment ils ont obtenus leur ontologie, ou ils affirment utiliser une méthode qui ne permet pas de justifier les choix de conception.

CHAPITRE 3 ATOM : UNE MÉTHODE DE DÉVELOPPEMENT D'ONTOLOGIES APPLIQUÉES À LA SÉCURITÉ

Dans ce chapitre, nous répondons à notre première question de recherche en présentant ATOM ("Abstractions Translation Ontology Method"), la solution que nous proposons aux difficultés présentées par la création d'ontologies. Nous commençons par faire une courte présentation du processus suivi lors de la création de la méthode ainsi qu'un court historique de celui-ci. Nous exposons ensuite le contexte d'utilisation de l'ontologie, soit le système expert permettant de prendre des décisions relatives à la sécurité informatique d'un système étudié. Puis, nous présentons les idées maîtresses sur lesquelles ATOM est basée. Finalement, nous expliquons la méthode proposée et l'illustrons par un exemple de son utilisation.

3.1 Processus de développement de la méthode

Comme présenté dans l'introduction, nous avons suivi l'intuition de l'utilisation de l'ontologie pour aborder certains problèmes de représentation de la connaissance reliés à la sécurité informatique. Notre contexte d'utilisation de l'ontologie est le système de détection d'intrusion. Nous étendons cependant ce contexte au système expert qui permet de prendre des décisions relatives à la sécurité au sujet d'un environnement étudié, définition qui inclut le SDI. Ce contexte nous a menés à notre problématique : l'inexistence d'une méthode de développement ontologique appliquée à ce contexte. Nous avons donc voulu proposer une méthode de développement comme solution à cette problématique et notre méthodologie a été la conséquence directe de ce but.

Le processus de la création d'une méthode de développement est très itératif. Notre méthodologie a donc été de nature inductive et expérimentale. Notre point de départ a été des rencontres d'experts des domaines des environnements informatiques et du contrôle aérien ayant pour but la création de lexiques des concepts pertinents. Par la suite, nous avons eu plusieurs rencontres subséquentes où les ontologies étaient discutées et développées. Ce processus était inspiré des méthodes existantes. Au fil du processus, nous avons constaté plusieurs lacunes reliées à ces méthodes. Le contexte d'utilisation de l'ontologie n'était pas clair. Il était souvent difficile de prendre des décisions de modélisation raisonnées et justifiables. De plus, il n'était pas suffisant d'avoir seulement des listes de mots reliés au domaine à la manière d'un champ lexical avec leur définition logique. Il était nécessaire de mettre en relation les entrées et les sorties attendues du système et la participation concrète de l'ontologie dans ce processus. C'est pour répondre à ces problématiques qu'il a fallu développer une méthode

plus particulière à notre contexte.

Dans la première étape, nous avons tout d'abord explicité clairement de quelle manière nous allions utiliser l'ontologie ou autrement dit, de quelle manière le système expert de détection d'intrusion allait l'utiliser. Puis, nous avons choisi des manières de représenter l'information qui permettraient une spécification du système expert propice à l'implémentation. La dernière tâche de cette étape a été de concevoir les grandes lignes d'un processus de manière à pouvoir l'utiliser et l'améliorer.

La deuxième étape a été celle qui s'est étendue sur la plus longue période. En effet, à partir de l'ébauche d'une méthode, nous avons travaillé sur la modélisation des ontologies des environnements informatiques et du contrôle aérien. Au fur et à mesure du développement, nous avons amélioré la méthode en répondant aux problématiques rencontrées et en essayant de simplifier et de systématiser le processus, tout en gardant à l'esprit les buts que nous voulions atteindre avec cette méthode ainsi que nos questions de recherche. La conséquence de ce processus d'améliorations successives a été que nous avons dû recommencer quelques fois les modélisations et nous avons dû souvent revenir en arrière dans le processus.

La troisième et dernière étape a été de compiler les résultats obtenus en explicitant clairement la méthode et en obtenant des données du processus, c'est-à-dire le temps de développement, le nombre de personnes impliquées et les différences obtenues entre la modélisation des systèmes informatiques et celle du contrôle aérien.

À travers le déroulement de notre processus de recherche, particulièrement durant la deuxième étape, nous avons progressivement défini ce que nous cherchions à obtenir, c'est-à-dire quelles étaient les visées de la méthode, de manière à pouvoir répondre à notre deuxième question de recherche. Nous avons décidé qu'elle devait répondre à cinq exigences.

1. Nous voulions que la méthode produise des documents de spécification clairs permettant l'implémentation concrète d'un système expert. Pour ce faire, il fallait s'assurer que le plus d'éléments possibles soient documentés afin de réduire les zones d'ombre.
2. La méthode devait permettre d'éviter de perdre trop de temps sur la modélisation. En effet, notre expérience nous a montré qu'il est souvent possible de s'éterniser sur des modélisations sans qu'il n'y ait de métriques concrètes permettant la comparaison de celles-ci.
3. Elle devait permettre de diriger le processus de développement, c'est-à-dire qu'elle devait expliciter étapes par étapes le processus à accomplir pour encoder la connaissance de l'expert et proposer des outils et des procédures.
4. L'ontologie produite par la méthode devait être d'une qualité suffisante pour être

partagée et réutilisée dans d'autres contextes.

5. Le processus complet devait être d'une durée raisonnable pour son application en industrie.

3.2 Contexte : le système expert

Ce qui nous distingue de la plupart de la recherche sur les méthodes de modélisation ontologique est l'utilisation de l'ontologie dans un système expert. En effet, dans ce contexte, l'ontologie sert uniquement de modèle de stockage de l'information sous la forme de graphe orienté. Dans le langage des bases de données, on utilise le terme "schéma" pour parler de ce modèle. On qualifiera le système de stockage en tant que tel de "base de données ontologique" ou simplement de "base de données". Cette base de données s'inscrit dans un système plus large qu'on pourrait qualifier de système de connaissances ou basé sur les connaissances, parfois aussi appelé "système expert" lorsqu'il a pour but de répondre à des questions relatives à un domaine précis. Ce système sert de contexte à l'ontologie et permet de répondre à diverses questions de modélisation.

3.2.1 Architecture du système

Le système qui sert de contexte aux ontologies développées est composé d'agents interagissant avec le monde extérieur, qu'on appellera "senseurs", d'agents créant de la nouvelle information en se basant sur l'information présente dans la base de données, qu'on appellera "traducteurs", et d'agents permettant de questionner la base de données, qu'on appellera "interfaces". Pour faire un parallèle avec le corps humain, on peut considérer que les senseurs sont les yeux du système, que les traducteurs sont son cerveau et que les interfaces sont la parole qui lui permet de communiquer avec le monde extérieur. Aussi, les senseurs ne devraient en principe faire aucun traitement de l'information et devraient acheminer simplement l'information au système telle qu'elle est observée, les traducteurs ne devraient avoir un accès qu'aux informations internes telles que transmises par les senseurs et les interfaces ne devraient consister qu'en la formulation de l'état interne de connaissance pour le monde extérieur sans plus de traitement. Une des tâches importantes à considérer lors du développement d'une ontologie est son contexte d'utilisation par rapport à ces différents types d'agents. En effet, nous nous attendons à ce que la sémantique des termes de l'ontologie découle d'une compréhension des termes par les développeurs des agents logiciels interagissant avec celle-ci. Une illustration de cette architecture est présentée à la Figure 3.1.

Il est utile ici d'introduire le terme "pilote" relatif aux agents senseurs. En effet, la plupart

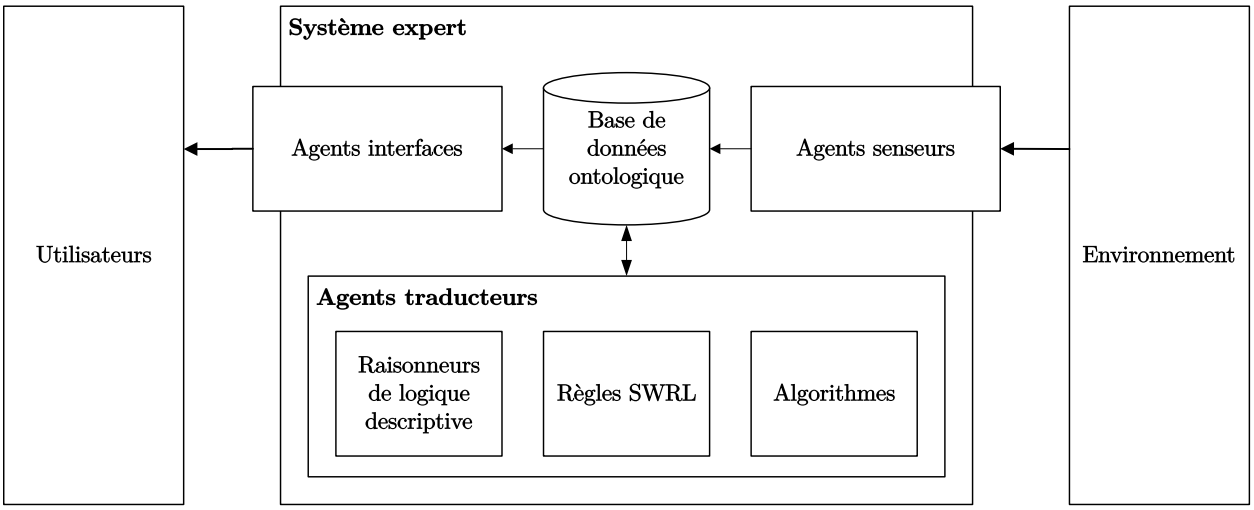


Figure 3.1 Architecture du système expert

du temps, du point de vue du système expert, les senseurs sont préexistants. Des exemples de senseurs sont les systèmes d'exploitation, les SDI sur réseau ou sur hôte, les antivirus, les pare-feu, les points d'accès réseau comme les "routers", "switch", etc. Ces composantes sont donc externes au système expert et l'agent senseur se trouve en fait être un "pilote" qui permet au système expert de s'interfacer avec les senseurs qui génèrent de l'information sur le monde extérieur surveillé. Ainsi, conceptuellement, les termes "pilote" et "senseur" sont semblables et seront souvent utilisés de manière interchangeable, car ils font tous les deux référence à l'entrée d'information brute dans le système expert. Cependant, du point de vue plus formel de spécification, ils sont différents.

3.2.2 Exemple général d'utilisation

Pour illustrer l'implication respective des différents types d'agents, nous présentons maintenant un exemple relatif au contrôle aérien. Supposons que nous cherchons à savoir si la trajectoire en vol d'un véhicule aérien a dévié de son plan de vol. La granularité de l'information de la trajectoire du plan de vol et du chemin réel peut ne pas être la même. Cela nécessite donc un calcul assez complexe qui ne peut être effectué simplement dans une requête à la base de données. Nous pouvons alors imaginer un agent "traducteur" qui compare constamment les données de vol avec le plan de vol et qui détecte la déviation de la trajectoire d'un véhicule, à l'aide de règles d'interpolation de trajectoire. Cet agent pourra alors mettre à jour la base de données en générant un événement de déviation de trajectoire relié aux points répondant à ce critère. D'un autre côté, nous pouvons imaginer un agent interface

demandant la présence de nouveaux événements à la base de données et qui met à jour une interface utilisateur pour alerter un opérateur de la déviation de trajectoire détectée par un agent traducteur, par exemple. Finalement, nous voyons que l'agent traducteur a effectué son calcul sur des points de position représentés uniformément. Cependant, on sait qu'en contrôle aérien, ces points de données peuvent provenir de trois sources, c'est-à-dire des radars primaires, secondaires et des antennes ADS-B. Le senseur qui génère ces informations devrait alors obligatoirement cheminer ces données à un autre traducteur qui pourrait alors fusionner ces trois types d'information en une seule, de manière à faciliter le travail du traducteur qui détecte les déviations de trajectoires. Le senseur pourrait aussi sauvegarder les données brutes dans la base de données au besoin.

Il faut noter qu'il est possible qu'un agent entre dans plus d'une catégorie. Par exemple, un agent interface pourrait poser une question complexe à la base de données afin de détecter un événement et avertir un utilisateur en faisant potentiellement la requête. Cet algorithme ne mettra pas nécessairement à jour la base de données avec un événement, mais fera office de traducteur et d'interface.

3.2.3 Indices d'implémentation

Afin de mieux comprendre le système expert proposé, on donne dans cette section des pistes d'implémentation sans entrer trop dans les détails. Pour illustrer comment on implémenterait ce système, prenons l'exemple de la détection d'une attaque quelconque. Supposons que pour détecter cette attaque, le système doit savoir si une charge utile a été envoyée à un serveur et si ce serveur est vulnérable à cette attaque. Si ces deux conditions sont réunies, le système pourra alors alerter l'utilisateur qu'un serveur du réseau est attaqué.

Le point de départ du système expert est la base de données ontologique. Ces logiciels offrent des mécanismes qui permettent de créer une base de données basée sur une ontologie donnée, puis d'en faire la gestion, c'est-à-dire d'y ajouter de la nouvelle information, d'en supprimer, de la modifier et de la questionner.

Prenons ensuite le cas des senseurs. Pour atteindre les buts fixés du système expert, l'ontologie doit modéliser deux types d'information. Nous qualifions le premier type de connaissance préalable. Ce type d'information comprend la connaissance générale du domaine d'application et l'information qui est peu sujette au changement. Dans notre exemple, les serveurs de l'environnement, leurs services et leurs modules respectifs sont une connaissance préalable. Nous qualifions le deuxième type d'information de connaissance dynamique. Cette information est celle qui est constamment générée et avec laquelle les événements d'intérêts sont principalement générés. Elle permet de passer de l'information générale d'un domaine de

connaissance à l'information spécifique d'un contexte étudié. Dans notre exemple, la détection d'une requête contenant une charge utile à un moment donné est une connaissance dynamique. Bien que dans les deux cas ce soit la tâche des senseurs de récupérer l'information, la distinction est importante pour le processus de mise en service du système expert. En effet, la connaissance préalable sera fournie au système expert avant sa mise en service alors que la connaissance dynamique sera constamment ajoutée tout au long de la mise en service du système. Cela a un impact sur l'implémentation des senseurs. Dans le premier cas, le senseur devra être exécuté une seule fois avant la mise en service pour remplir la base de données. Par exemple, un senseur pourrait lire la liste des serveurs existants d'un CMS (Content Management System) et les entrer dans la base de données. Dans le deuxième cas, le senseur devra constamment écouter le canal d'information et ajouter au fur et à mesure la nouvelle information. Dans notre exemple, un senseur devra s'interfacer avec un SDI tel que "Snort" afin de mettre dynamiquement à jour la base de données lorsqu'une attaque est détectée. Dans ce qui suit, nous avons fait la distinction en précédant les identifiants des senseurs du premier cas d'un "C", pour "configurateur", et ceux du deuxième cas d'un "D".

Une fois l'information brute présente dans le système expert, les agents traducteurs pourront exécuter leur travail soient à des périodes de temps prédéfinies, soit en fonction d'une certaine quantité de nouvelle information, soit à l'instigation d'une demande effectuée par les senseurs dans des situations prédéfinies. Par exemple, si un senseur qui s'interface avec un SDI observe la présence d'une attaque, il pourra créer l'information dans la base de données ontologique, puis appeler la procédure d'un agent traducteur qui évaluera, en se basant sur l'information présente dans la base de données, si l'attaque est pertinente ou si elle est un faux positif. Le travail de ce traducteur peut être constitué de l'exécution du travail d'inférence d'un raisonneur de logique descriptive sur l'ontologie, de l'application de règles SWRL ou de tout autre algorithme qui lit l'information présente dans la base de données, tire des conclusions et l'actualise selon ces dernières. Dans tous les cas, cela équivaut à créer de nouvelles informations basées sur ce qui se trouve déjà dans la base de données. Dans notre cas, une règle SWRL reliera les événements d'attaque aux services instanciées dans le système. Par inférence en logique descriptive, les serveurs qui offrent un service vulnérable pourraient être classifiées dans la classe des serveurs vulnérables. Puis, un autre agent créera une instance d'attaque en se basant sur le fait que le service d'un serveur vulnérable a été ciblé par une attaque.

Finalement, un utilisateur exécutera une requête SPARQL donnée qui consistera à chercher la présence d'instances d'attaques détectées dans la base de données ontologique. En effet, la plupart des logiciels de bases de données ontologique offrent un service de requête SPARQL.

3.3 Fondation

ATOM se base principalement sur l'hypothèse suivante : le travail principal de l'expert en sécurité consiste à récupérer les multiples données générées par un système, puis à classifier, simplifier et connecter ses données de manière à pouvoir en tirer des conclusions. Ce travail lui permet de passer d'une représentation très brute de l'information à une représentation à plus haut niveau plus propice à l'utilisation. Le but du système expert est donc d'effectuer exactement ce type de traduction. Ce point est très important pour comprendre la méthode proposée. Autrement dit, nous cherchons à développer un système expert qui transforme l'information brute en une information plus abstraite, permettant des requêtes d'information à haut niveau d'abstraction. Les trois sections qui suivent présentent de quelle manière ce système expert y arrive.

3.3.1 Droite des niveaux d'abstraction

Comme nous l'avons présenté à la section 2.1.4, l'ontologie permet l'abstraction de l'information et c'est principalement cette caractéristique que l'on veut exploiter dans la conception d'un système expert utile aux travailleurs de la sécurité. Les données brutes qui décrivent l'environnement étudié peuvent se situer à différents niveaux d'abstraction. Par exemple, un analyste qui voudrait savoir si un média externe a été inséré dans une machine à un moment donné devrait chercher un événement de partition montée dans les fichiers journaux du système d'exploitation de cette machine. Cet événement pourrait ne pas avoir clairement l'étiquette de "partition montée", mais plutôt présenter un chiffre lui servant d'identifiant, en quel cas l'analyste devrait chercher dans la documentation du système la sémantique associée à cet identifiant. D'un autre côté, le senseur pourrait être beaucoup plus intelligent et faire un traitement de l'information brute de manière à pouvoir l'étiqueter à l'aide de concepts plus abstraits. Par exemple, un système de détection d'intrusion réseau, comme "Snort", analysera le contenu des paquets réseau pour tenter de déceler une séquence binaire précise, indiquant qu'un attaquant essaie d'utiliser la vulnérabilité "heartbleed" pour récupérer de l'information de manière illicite. Lorsqu'un tel cas est détecté, l'outil crée une alerte abstraite étiquetée à l'aide de la classe d'attaque, dans ce cas-ci, une attaque "heartbleed". Ici, l'analyste aura donc une information à un plus haut niveau d'abstraction que pour le cas de l'insertion d'un média externe. De la même manière, l'expert en sécurité se construira une représentation mentale avec différents niveaux de précision, suivant le cas étudié. Pour cette raison, il voudra parfois interroger le système à de très hauts niveaux d'abstraction, alors qu'à d'autres moments il voudra avoir une information plus concrète, ou plus proche de l'environnement étudié.

De ce qui précède découle l'idée maîtresse de ATOM. Elle consiste en une droite présentant le niveau d'abstraction de l'information. À une extrémité de cette droite, on retrouve l'information à son plus bas niveau d'abstraction et à l'autre, on retrouve l'information à son plus haut niveau d'abstraction. Entre ces deux extrêmes, on retrouve un très grand nombre de représentations transitionnelles. Pour illustrer cette idée, nous pouvons prendre l'idée d'une image humoristique, par exemple une caricature d'un politicien connu, s'affichant sur un écran d'ordinateur. Si on se place à un niveau très bas d'abstraction de cette information, on retrouvera un très grand nombre d'atomes de différentes natures organisés d'une certaine manière dans l'espace. À un niveau supérieur, on verra que ces atomes forment en fait la mémoire d'affichage d'un ordinateur. En montant la droite d'abstraction, on pourra découvrir à un moment l'ensemble de pixels s'affichant à l'écran et formant l'image. En montant encore plus haut, on retrouvera le visage du personnage en question. On reconnaîtra les différentes parties de sa physionomie, son nez, sa bouche, ses yeux, etc. Finalement, on associera cette image à un concept très abstrait qu'est la personne avec son nom, ses fonctions et toutes autres connaissances qui lui sont associées et on comprendra le ton humoristique de l'information véhiculée. On illustre cet exemple à la Figure 3.2. Un utilisateur peut demander d'accéder à cette information à différents niveaux d'abstraction, selon le besoin informationnel. Par exemple, un être humain qui parle à un autre pourra lui demander qui est la personne caricaturée, faisant ainsi référence à un très haut niveau d'abstraction de l'information. En admettant que l'image en question soit sur un serveur connecté à l'internet, un client pourra requérir l'image à ce serveur. Ce niveau de questionnement sera à un niveau d'abstraction beaucoup plus bas et consistera, par exemple, à l'information binaire structurée selon un certain format permettant l'affichage de l'image sur un écran.

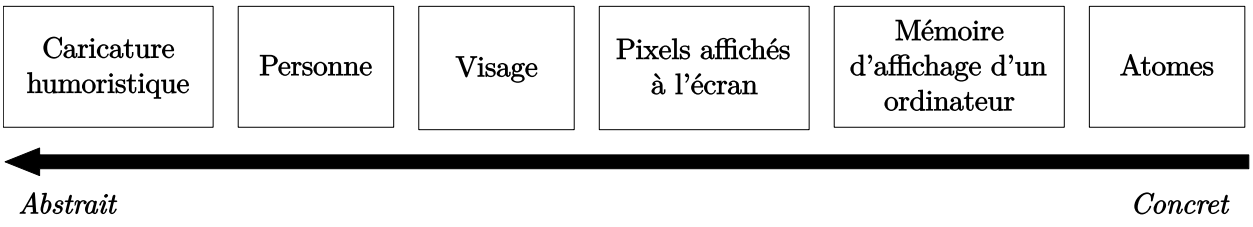


Figure 3.2 Exemple d'une droite d'abstraction

3.3.2 Traductions des niveaux d'abstraction

La vision de la représentation de l'information sur une droite de niveaux d'abstraction est certes simplifiée, mais elle est suffisante pour notre but. En effet, différents senseurs génèrent

l'information à différents niveaux d'abstraction et dans un contexte donné, les analystes ou les agents logiciels sont intéressés à un niveau d'abstraction spécifique. Ainsi, le rôle du système expert est de récupérer l'information des senseurs et d'exécuter diverses procédures pour traduire l'information d'un niveau d'abstraction à l'autre. Ces traductions peuvent nécessiter différentes connaissances, encodées de différentes manières. Par exemple, une traduction peut être une simple inférence de classification en raison d'une relation de sous-classe, comme par exemple une "requête HTTP" classifiée comme étant une simple "requête" à cause de la relation de sous-classe entre "requête HTTP" et "requête", alors qu'une autre peut nécessiter des calculs complexes. Le but général est de passer des représentations à très bas niveaux d'abstraction, couramment utilisées par les machines, vers des représentations à plus haut niveau d'abstraction, couramment utilisées par les humains.

Dans l'approche que nous étudions, il existe principalement trois types de traduction : la traduction par inférence logique, la traduction par application de règles et la traduction algorithmique. Cette classification découle des outils actuellement utilisables dans le contexte de la représentation de la connaissance à l'aide d'une ontologie. Le raisonneur peut exécuter des inférences en se basant sur les axiomes définis en logique comme présenté dans la section 2.1.3. Cela peut aller de simples relations de classification à des définitions plus complexes basées sur des chaînes de propriétés, selon le vocabulaire utilisé. La logique descriptive a cependant des limitations qui peuvent être comblées en utilisant des outils raisonnant à base de règles comme SWRL ou l'élément "CONSTRUCT" de SPARQL. Par exemple, il n'est pas possible de l'utiliser pour la création de nouveaux axiomes dans l'ontologie. Finalement, dans des cas plus complexes, on utilise un langage de programmation classique pour récupérer l'information, la traiter et mettre à jour la base de données par la suite. On peut par exemple utiliser un algorithme écrit en Java dans le cas où l'on veut exécuter un calcul complexe à partir de l'information présente dans la base de données. Comme exemple de tels calculs, on peut vouloir exécuter des calculs mathématiques complexes comme l'intersection entre des plans dans l'espace ou calculer des événements à partir d'information nécessitant un système externe. On peut aussi vouloir faire appel à des modèles construits à l'aide de méthodes quelconques d'apprentissage machine pour prendre des décisions. Par exemple, si une information à très bas niveau est une série de pixels formant une image, on peut utiliser un modèle qui sait comment catégoriser cette image. Dans tous les cas, ces outils ont pour but de tirer des conclusions, c'est-à-dire de créer des nouvelles informations dans la base de données, à partir de l'information s'y trouvant au préalable. Le but dans notre contexte est toujours de traduire l'information d'un niveau d'abstraction à un autre.

3.3.3 Le requis comme moteur de développement

Nous avons parlé de la droite des niveaux d'abstraction et du besoin de traduction pour se déplacer sur celle-ci. Nous abordons maintenant le moteur qui contrôle l'effort de localisation sur cette droite et donc, par la même occasion, l'effort de traduction. Cet élément est important, car comme nous l'avons spécifié plus tôt, il existe une infinité de manières de représenter l'information. Il faut donc contraindre la représentation en définissant exactement de quelle manière le système sera interrogé. Pour ce faire, nous devons définir une séquence de requêtes allant de la question la plus primitive en montant vers la question la plus abstraite. Ces deux requêtes font office de bornes inférieure et supérieure permettant de limiter le champ d'action du système expert. L'effort minimal de traduction à accomplir est donc celui qui permet au système de répondre à la requête au plus haut niveau d'abstraction défini. Cette dernière sert de spécification fonctionnelle du système alors que la requête à plus bas niveau sert en fait de documentation de l'information brute disponible au processus de raisonnement, ou autrement dit, de documentation des pilotes permettant l'interfaçage avec l'environnement étudié. Par exemple, une requête de haut niveau est : "Est-ce qu'un média externe a été inséré ?". La requête de bas niveau qui y est associée est : "Est-ce qu'il existe un événement de système d'exploitation Windows avec une valeur d'identification X ?" La spécification fonctionnelle du système dans cet exemple consiste à permettre à son utilisateur de savoir si un média externe a été inséré à un moment donné dans une machine. La documentation du pilote indique qu'il doit lire les événements Windows et y chercher un numéro d'identification particulier. Dans ce cas-ci, nous avons une correspondance un à un. Mais, il arrive souvent que la requête de haut niveau soit associée à un ensemble de questions de bas niveau. Nous disons alors que cette requête a plusieurs "branches". Par exemple, pour répondre à la question de haut niveau, à savoir si un administrateur s'est connecté sur une machine quelconque, il faudra à plus bas niveau savoir quels utilisateurs se sont connectés sur cette machine et quels utilisateurs sont administrateurs. Ces deux informations devront ensuite être combinées pour permettre de répondre à la première question.

On doit maintenant traiter la question des représentations intermédiaires de cette séquence de requêtes. La création de celles-ci dépend de trois facteurs : l'intuition de l'expert impliqué dans le processus de modélisation, l'effort nécessaire à la conception des traductions et les besoins de fusion des ontologies. Pour comprendre ces facteurs, il faut tout d'abord comprendre que la droite d'abstraction est en fait documentée par une séquence de requêtes. Il n'existe pas d'autre formalisme pour spécifier ce genre de hiérarchie. Nous présentons maintenant ces facteurs plus en détail. Premièrement, lors de la conception du système, l'expert du domaine peut avoir des intuitions quant aux requêtes intermédiaires en se basant sur sa connaissance

du domaine et son expérience dans l'accomplissement de la tâche pour laquelle le système expert est conçu. Il peut par exemple trouver pertinent l'ajout de concepts intermédiaires qui pourraient l'intéresser dans certains cas ou qu'il pense pouvoir être utiles dans d'autres contextes. Par exemple, dans le contexte de la détection de trafic potentiellement malveillant sur un réseau, une branche de notre droite d'abstraction a une requête de bas niveau qui consiste à obtenir les requêtes réseau sur des ports en particulier. La requête de plus haut niveau utilise cette branche pour demander si du trafic malveillant circule. Lors de ce travail, l'expert a jugé bon d'ajouter les concepts intermédiaires de protocoles HTTP, DNS, etc. Ainsi, une requête intermédiaire a été créée entre la requête de trafic réseau sur des ports donnés et celle du concept de "trafic malveillant", qui consiste à demander les requêtes dans un protocole X ou Y. Deuxièmement, une méthode très connue de la résolution de problèmes complexes est celle de "diviser pour régner". La plupart du temps, les requêtes plus abstraites peuvent être divisées en questions plus simples et plus concrètes. Plus exactement, cela signifie que pour permettre la réponse à une requête abstraite, le traducteur doit pouvoir répondre à une ou plusieurs questions plus concrètes. Dans ce cas-ci, les questions intermédiaires servent de documentation à la procédure de traduction, puisque chacune d'entre elle consiste en la documentation de ce que la procédure de traduction devra obtenir du système pour pouvoir mener à bien son travail. Troisièmement, la montée en abstraction implique souvent la fusion de plusieurs sources d'information. Certaines questions intermédiaires indiqueront donc simplement à quel moment une information doit être disponible. Par exemple, l'analyste en sécurité peut être intéressé seulement aux balayages de port lorsque ceux-ci sont faits sur un serveur. Pour répondre à cette interrogation, il faut que le système expert mette en relation les événements générés par "Snort" et les machines cibles de ces événements. Une fois ce travail de liaison fait, il suffit d'isoler seulement les événements dont la cible est un serveur. Ainsi, on fusionne de l'information événementielle et de l'information d'environnement.

3.4 Présentation de ATOM

Nous présentons maintenant ATOM, la méthode de développement d'ontologie que nous proposons à proprement dit en expliquant et en justifiant chacune de ses étapes. Puis, nous l'illustrons par un exemple relatif à la sécurité informatique.

3.4.1 Présentation générale

ATOM est résumée par la Figure 3.3. Elle est composée d'une première étape de spécification des besoins, puis de 4 étapes avec le développement de l'ontologie s'opérant au travers de celles-ci. La cinquième et dernière étape consiste en l'enrichissement de l'ontologie dé-

veloppée. Trois artefacts sont créés au fil de l'application de la méthode : un diagramme documentant le processus de traduction du système expert, une spécification générale du système comportant entre autres la description des pilotes interfaçant le système expert avec ses senseurs, les requêtes SPARQL et les règles utilisées pour transformer l'information et finalement, l'ontologie comme telle avec ses concepts, propriétés et règles. Nous présentons plus en détail chacun de ces éléments ci-dessous. Nous commençons cette présentation par les artefacts produits, puisque nous leur ferons référence lorsque nous présenterons par la suite les étapes de la méthode.

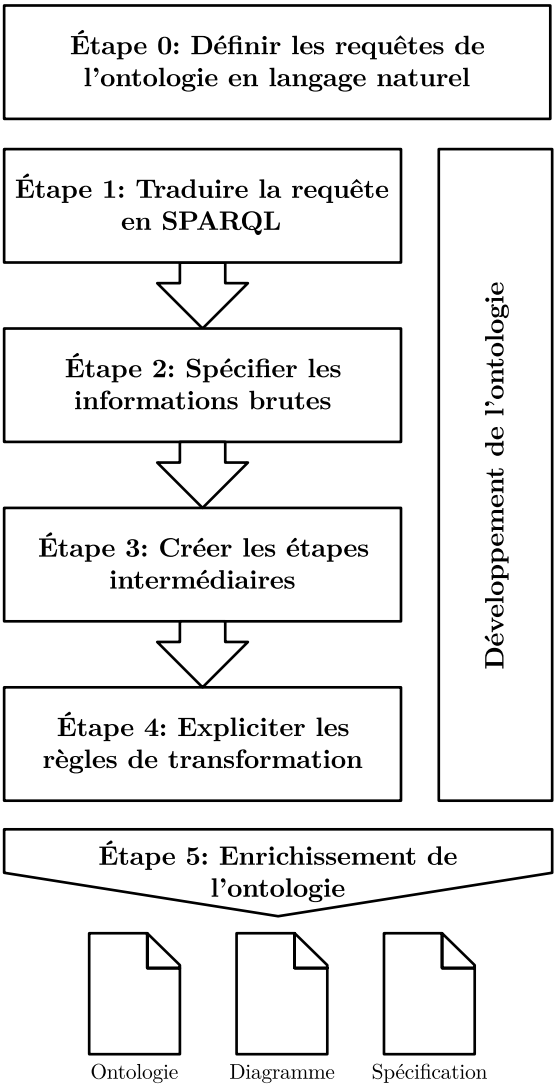


Figure 3.3 Résumé de la méthode ATOM

Artefacts produits

L'ontologie est stockée sous la forme d'un document OWL créé par Protege ou un autre outil de création d'ontologies. Il contient les axiomes écrits dans une syntaxe concrète au choix, par exemple "turtle". Ce document doit pouvoir être lu et modifié par les outils de gestion d'ontologies. Il contient les définitions des classes et propriétés de l'ontologie, des exemples d'instances et de leurs annotations.

Le diagramme de traductions est utilisé pour le développement de l'ontologie et documente le comportement du système expert et les questions auxquelles il doit pouvoir répondre. C'est dans ce document qu'on retrouve les droites de niveaux d'abstraction illustrées par des séquences de requêtes SPARQL. À un extrême, on retrouve le symbole des pilotes et configurateurs créant l'information brute dans le système et à l'autre, on retrouve la requête au plus haut niveau d'abstraction qui intéresse l'expert en sécurité. Entre les deux, on retrouve une série de requêtes et de règles de traduction qui permettent de passer d'une extrémité à l'autre.

La Figure 3.4 donne un exemple concret du formalisme utilisé dans le document.

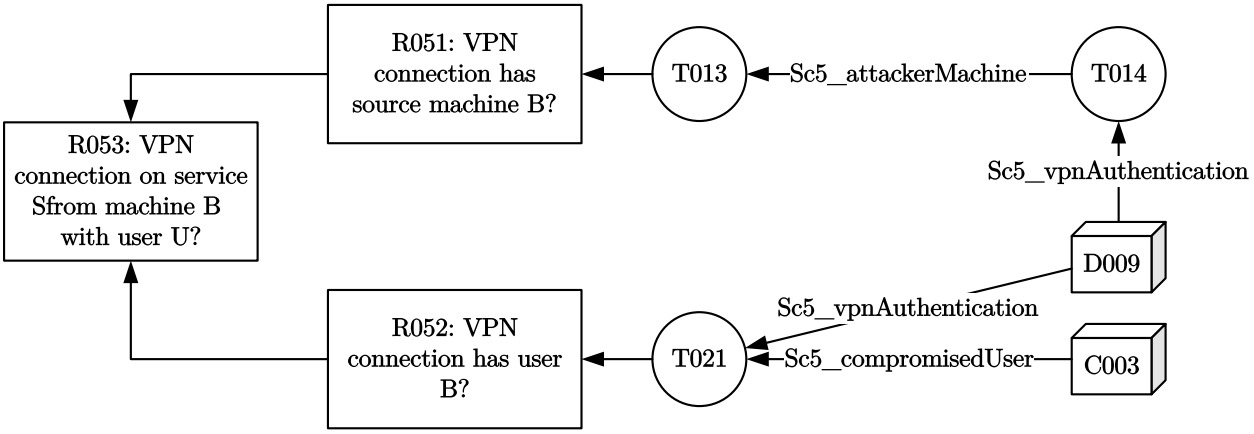


Figure 3.4 Exemple de diagramme de flot de traduction

Ici on a deux pilotes "D009" et "C003" s'interfaçant avec des senseurs quelconques qui créent respectivement les instances "Sc5_vpnAuthentication" et "Sc5_compromisedUser". La requête qu'on veut demander au système est "R053". Pour ce faire, nous avons jugé pertinent deux requêtes intermédiaires "R051" et "R052". Puis, nous avons créé les règles de traduction "T013" et "T014" pour permettre de répondre à "R051" à partir de "D009" et "T021" pour permettre de répondre à "R052" à partir de "C003" et "D009". Nous pouvons noter

que la règle de traduction “T014” crée une instance “Sc5_attackerMachine” qui est nécessaire pour répondre à “R0051”. Notons aussi ici que les instances créées le sont simplement à titre d'exemple et servent à documenter plus en détail les pilotes et les règles de traduction. Une description plus détaillée des éléments utilisés pour ce document se trouve dans le Tableau 3.1.

Remarquons que pour la représentation des requêtes, nous écrivons dans les boîtes une courte description de la requête en langage naturel en plus de l'identifiant de la requête. Pour tous les autres éléments, nous indiquons seulement leur identifiant. Dans le cas des instances, cet identifiant fait directement référence à leur IRI dans l'ontologie. Dans le cas des requêtes, règles de traduction et pilotes, l'identifiant fait référence au document de spécification qui décrit plus en détail chaque élément.

Le document de spécification décrit les éléments introduits dans le diagramme de traductions de manière à permettre une implémentation du système expert. Il est divisé en trois sections, soit la spécification des requêtes, celle des règles de traduction et celle des pilotes. La spécification des pilotes est elle-même divisée en deux pour faire la distinction entre les pilotes dynamiques qui lisent les informations d'événements et ceux statiques qui lisent les informations d'environnement comme présenté à la section 3.2.3. L'information est simplement présentée dans des tableaux où l'on indique à gauche l'identifiant de l'élément et à droite ce qui le décrit.

Nous décrivons maintenant l'information présentée dans ce document, qui varie en fonction de l'élément documenté. Dans le cas des pilotes, on décrit en langage naturel avec quel senseur chacun s'interface, comment ils récupèrent l'information et les instances créées dans la base de données avec chacune de leur propriété. Par exemple, le pilote “D008” doit lire les fichiers journaux des serveurs web installés sur un ordinateur pour extraire toutes les requêtes HTTP auxquelles ils ont répondu. Il doit ensuite créer pour chaque requête une instance de type “HTTPRequest” en lui donnant les propriétés “hasDomain”, “hasPath” et “hasSourceIPAddress”. Dans le cas des requêtes, on indique simplement la requête SPARQL. Il est possible de copier la requête telle quelle et de la coller dans un outil d'interrogation SPARQL. Les requêtes SPARQL étant assez explicites et étant donné qu'une formulation en langage naturel de la question se trouve déjà dans le diagramme de traduction, nous avons décidé de n'ajouter aucune autre information dans le document de spécification. Finalement, les règles de traduction ont une colonne supplémentaire pour indiquer de quel type il s'agit. Pour l'instant, elles sont de trois types : en logique descriptive, SWRL et algorithmique. Dans le cas des traductions explicitées en logique descriptive et des traductions algorithmiques, on la décrit simplement en langage naturel. Un exemple de règle algorithmique est celle qui vérifie, pour

chaque “Machine” présente dans le système, si elle se trouve dans la liste de “Google Safe Browsing”. On va parfois expliciter plus en détail le fonctionnement de l’algorithme, mais, dans ce cas-ci, il est laissé au développeur le travail de trouver quelle propriété des machines utiliser pour la vérification. Cela contribue au processus itératif de développement du système. En effet, un programmeur pourrait déterminer qu’il n’a pas l’information nécessaire pour accomplir sa tâche et qu’il faudrait alors récupérer plus d’information. Cela impliquerait une modification du pilote “C001” qui crée les instances de machine dans le système. Un exemple de règle exploitant le raisonneur de logique descriptive concerne les généralisations de classe possibles en raison des relations de sous-classe. On écrira simplement dans ce cas qu’une instance de tel type est aussi de tel autre type en raison de telle relation de sous-classe. Par exemple, la règle “T007” affirme que toutes les instances de type “MountedPartition” sont aussi de type “Event” parce que “Event” est une superclasse de “MountedPartition”. Finalement, on documente les règles de traduction utilisant SWRL en écrivant directement la règle elle-même. Encore une fois, nous n’avons pas jugé nécessaire d’ajouter une description plus explicite à cause de la clarté des règles SWRL et de la facilité à comprendre ce qu’elles font. Par exemple, pour ajouter le protocole HTTP à toutes les requêtes réseau qui ont comme protocole TCP et qui sont sur le port 80, il suffit d’écrire l’énoncé 3.1.

$$\begin{aligned}&\texttt{NetworkRequest(?event)} \wedge \texttt{hasDestinationPort(?event, 80)} \wedge \\ &\texttt{hasProtocol(?event, TCP)} \rightarrow \texttt{hasProtocol(?event, HTTP)}\end{aligned} \tag{3.1}$$

Étape 0 : Définir les requêtes de l’ontologie en langage naturel

Comme on peut le lire dans le guide du corpus de connaissance en génie logiciel (SWEBOK), “les chercheurs et les professionnels de l’industrie reconnaissent largement que les projets logiciels sont extrêmement vulnérables lorsque les activités liées aux exigences sont mal réalisées.” (Bourque et al., 2014) Par exigences, ils entendent “les besoins et contraintes d’un produit logiciel qui contribue à la solution d’un problème du monde réel.” Le développement d’une ontologie n’échappe pas à cette règle. Il est nécessaire de formuler explicitement ce qu’on attend de l’ontologie avant de la développer. Étant donné que l’ontologie définit une manière de stocker l’information, ses exigences fonctionnelles auront toujours pour thème principal la requête, c’est-à-dire principalement les manières de l’interroger et les questions auxquelles elle devrait pouvoir répondre. L’étape initiale du développement est donc une formulation en langage naturel la plus exhaustive possible des questions que l’on veut poser à l’ontologie. On étiquette cette étape par le chiffre 0, car elle précède le développement de l’ontologie. Elle

consiste en la formulation du problème et peut être faite complètement à part, contrairement aux étapes subséquentes qui sont faites en parallèle avec le développement de l'ontologie.

Cette étape est importante, car un piège guette le développeur d'ontologie inexpérimenté : celui de viser la création de l'ontologie universelle de son domaine, c'est-à-dire l'ontologie qui modélise le mieux possible son domaine d'application avec peu d'égards immédiats pour son utilisation concrète. Des philosophes se sont penchés dans le passé sur un tel projet et ils ont conclu qu'"il conviendrait [...] de renoncer à l'idée même d'un "langage optimal"." (Monnin, 2015), c'est-à-dire qu'il n'y a pas de manière universelle pour représenter la connaissance qui soit toujours optimale, peu importe la situation. Le contexte a un impact majeur sur toute communication, modélisation ou tentative de représentation de l'information. Ce piège est dangereux, car il implique potentiellement beaucoup d'efforts dépensés inutilement. Par exemple, on pourra discuter des heures sur la définition d'un concept parce que son utilisation n'est pas bien définie et qu'on essaie d'être le plus général possible. On risque alors de tomber dans des discussions philosophiques de définitions et de langage. Cela peut être souhaitable dans le cadre de recherches en philosophie, mais ce ne l'est pas dans le cadre de l'ingénierie d'un système expert.

Le résultat de cette étape est une liste de questions écrites en langage naturel. Ses acteurs doivent être l'utilisateur final du système, l'expert du domaine et l'ontologiste.

Étape 1 : Traduire les requêtes en SPARQL

C'est à cette étape que le développement de l'ontologie commence. Pour chaque requête formulée à l'étape 0, on exécute les étapes 1 à 4, jusqu'à ce que toutes les requêtes aient été traitées. Concernant cette étape en particulier, de manière à ce que l'ontologie présente des modélisations se rapprochant le plus possible du langage de ses utilisateurs, on traduit les requêtes en SPARQL en s'imaginant de quelle manière on voudrait interroger la base de données. Une première formulation devrait être faite en se basant entièrement sur l'intuition de l'expert. On peut ensuite extraire les concepts et les relations de cette requête et les analyser à la lumière de ce qui est déjà présent dans l'ontologie. Pour chaque idée formulée dans la requête, on peut se trouver dans trois situations. La première situation survient lorsque le terme existe dans l'ontologie. Ce cas est le plus souhaitable, car on applique ainsi le principe de réutilisation et il empêche l'augmentation de la complexité de l'ontologie. La deuxième situation est lorsqu'un concept similaire existe dans l'ontologie. Dans ce cas, on doit réfléchir sur la pertinence d'introduire un nouveau concept dans l'ontologie. Est-ce que l'augmentation de complexité de l'ontologie est nécessaire ou est-ce qu'il serait préférable d'utiliser un concept déjà existant ? Si on choisit cette dernière possibilité, on doit alors

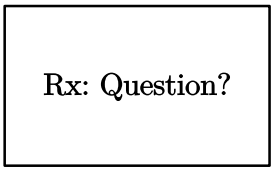
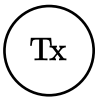
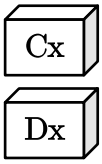
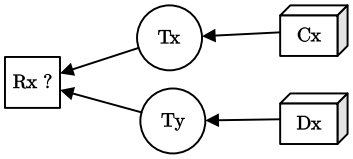
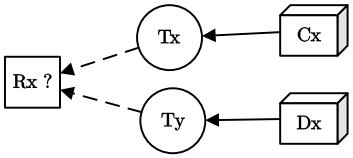
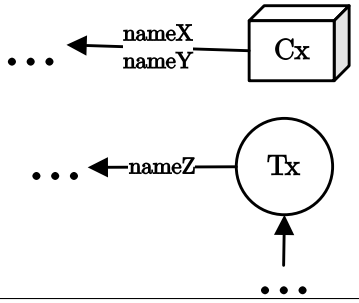
changer la requête pour refléter le concept préexistant. Finalement la troisième situation est lorsque le concept est totalement nouveau et qu'il faut l'ajouter obligatoirement dans l'ontologie. On s'attend à ce qu'au début, la plupart des situations soient de ce type, mais que, plus on avance dans le travail de modélisation, plus on obtienne le premier cas.

Cette démarche permet un développement de l'ontologie entièrement guidé par les requis. C'est le but que nous recherchons, comme nous l'avons expliqué dans la section 3.3.3. En effet, expliciter les requêtes SPARQL suivant l'intuition de l'utilisateur nous force à suivre son langage ou sa manière de s'exprimer. Cela est d'autant plus important qu'une même réalité peut être définie de manière différente en fonction des buts et intérêts du modélisateur. Ce fait est très bien décrit par Nelson Goodman, qui donne l'exemple d'un bagage dans un aéroport : "The spectator notice shape, size, color, material and even make of luggage, the pilot is more concerned with weight, and the passenger with destination and ownership. Which pieces of baggage are more alike than others depends not only upon what properties they share, but upon who makes the comparison, and when." (Goodman, 1972)

Cela permet aussi de résoudre certains problèmes courants dans la conception ontologique. Par exemple, il arrive souvent lors de la modélisation qu'on se demande si un concept doit être créé en tant que classe ou en tant qu'instance. Ici, la requête pourra requérir les objets d'une classe définie ou bien les objets ayant telle ou telle propriété. Cela se traduira par une modélisation sous la forme d'une classe dans le premier cas et sous la forme d'une instance dans le deuxième cas. Bien entendu, les deux modélisations sont valides, mais on privilégie celle qui se rapproche du langage de l'utilisateur. Un exemple concret de ce problème concerne les "requêtes HTTP". Est-ce que cet élément doit être modélisé comme une instance d'une classe "RequêteHTTP" ou bien doit-il être modélisé comme une instance de la classe "Requête" à laquelle est rattachée la propriété "aProtocole" avec pour valeur "HTTP" ? Étant donné que notre langage ne possède pas de terme distinct pour signifier la "requête HTTP", mais qu'il utilise plutôt la stratégie de composition du mot du concept général, "requête", et de la valeur de la propriété, "HTTP", cela nous indique que la deuxième option est à prioriser dans ce cas.

Les résultats de cette étape pour chaque question formulée à l'étape 0 sont une nouvelle entrée dans le diagramme de traduction, sa requête SPARQL dans le document de spécification et la création des nouveaux concepts et propriétés introduits par la requête dans l'ontologie. Ses acteurs sont au minimum l'expert du domaine et l'ontologiste.

Tableau 3.1 Légende du diagramme de traductions

Élément	Symboles	Description
Requête		Blocs de vérification d'états. Ils indiquent comment la base de données de triplets peut être interrogée à ce moment par une requête SPARQL. Ils peuvent être utilisés dans une machine à états par le système expert.
Règle de traduction		Règles de traduction qui doivent être exécutées dans une branche donnée. Elles peuvent être une règle SWRL, une inférence en logique descriptive ou un algorithme externe qui met à jour la base de données à partir d'information interne.
Pilotes		Pilotes ou configurateurs qui créent les données au plus bas niveau d'abstraction dans la base de données. Ils représentent l'interface avec l'environnement externe.
Préconditions		Pour une boîte donnée, les flèches signifient que pour qu'elle soit exécutable, tout ce qui lui est connecté par des flèches doit être d'abord exécuté. Par exemple, pour pouvoir demander Rx, Cx, Tx, Dx et Ty doivent être exécutées.
Préconditions alternatives		Lorsque les flèches sont pointillées, cela signifie qu'une seule des branches doit obligatoirement être exécutée. Par exemple, ici, pour pouvoir demander Rx, on doit soit exécuter Cx suivi de Tx, soit Dx suivi de Ty, soit les deux.
Exemples d'instances		Les étiquettes sur les flèches sont les IRIs des instances dans l'ontologie qui exemplifient ce que le configurateur/pilote/règle de traduction crée. Par exemple, ici, le configurateur Cx crée deux instances nommées "nameX" et "nameY" et la règle de traduction Tx crée une instance nommée "nameZ".

Étape 2 : Spécifier les informations brutes

L'étape suivante est spécifique à notre contexte d'utilisation de l'ontologie. En effet, l'idée générale du système expert est de traduire et de fusionner l'information obtenue des divers senseurs en un langage plus abstrait propre à l'interrogation par un utilisateur. Ainsi, une fois que nous savons ce qui sera demandé au système, il faut déterminer les sources d'information qui permettront d'y répondre ainsi que le format de cette information.

Comme nous l'avons spécifié à la section 3.2.1, nous appelons ces sources d'information des "senseurs" et ils peuvent être n'importe quel outil qui permet de récupérer de l'information sur l'environnement étudié par le système : antivirus, système d'exploitation, serveurs, etc. Du point de vue du concepteur, deux choses doivent être spécifiées : comment récupérer l'information et comment la stocker. Un expert en sécurité pourra répertorier les sources qu'il utiliserait selon les questions produites à l'étape 1. Le travail est alors d'étudier ces sources, les méthodes d'interfaçage avec celles-ci et les formats de l'information disponible, de manière à pouvoir spécifier le travail de récupération des pilotes du système expert. L'information récupérée par les pilotes des senseurs pourra ensuite être stockée telle quelle dans la base de données ou bien être traitée par les pilotes. La règle d'or ici est de limiter la logique dans les pilotes étant donné qu'ils font uniquement office d'yeux pour le système expert. Ainsi, il est préférable de stocker l'information telle quelle en utilisant la terminologie de bas niveau du langage naturel. Prenons l'exemple du pilote s'interfaçant avec le gestionnaire d'événements du système d'exploitation "Windows". Ces événements ont entre autres pour propriété un numéro d'identification de type qui permet à l'administrateur de savoir à quoi correspond concrètement cet événement, par exemple à l'insertion d'un média externe. Ce que nous affirmons par la limitation de la logique dans les pilotes est qu'il est préférable de stocker cet événement comme instance d'une classe "ÉvénementWindows" ayant pour propriété l'identifiant, plutôt que de déterminer en quoi consiste cet événement en se basant sur son identifiant et créer une instance, par exemple, de type "InsertionMediaExterne". En priorisant la première option, on "limite la logique" dans le pilote et on laisse le système expert faire ce raisonnement. Les résultats de ce travail de conception sont stockés en langage naturel dans le document de spécification. Pour chaque pilote, on associe une description en langage naturel, un identifiant unique et des instances exemplifiant son travail. Dans le diagramme, les instances servant d'exemple sont ajoutées comme étiquettes des flèches sortantes des boîtes représentant les senseurs. Cet exercice permet de faciliter la compréhensibilité, la testabilité, la réutilisabilité et la maintenance de l'ontologie et du système expert. Par compréhensibilité, nous voulons dire qu'un développeur de pilote pourra regarder tous les exemples générés par ce pilote pour mieux comprendre sa spécification. Par testabilité, nous voulons dire qu'étant

donné la présence des informations brutes dans l'ontologie telles qu'elles devraient avoir été générées par les senseurs, on peut utiliser les divers outils à notre disposition, par exemple les raisonneurs logiques, afin de tester concrètement les résultats, et ce, sans avoir besoin de l'implémentation complète des pilotes. On pourrait même imaginer un testeur automatique qui s'attend à avoir tel ou tel résultat à une requête en fonction des instances prépeuplées dans l'ontologie. Ainsi, une modification dans les règles ou dans l'ontologie pourrait être facilement validée. Le concept de réutilisabilité et de maintenance est très relié à celui de la compréhensibilité. En effet, il est nécessaire de comprendre ce qui est généré ainsi que le processus de traduction afin de valider si la réutilisation ou la modification d'une composante est pertinente ou pas. La documentation exacte du point de départ informationnel clarifie cela.

Pour chaque requête de l'étape 0, on retrouve comme résultat de cette étape des nouvelles entrées dans le diagramme de traduction et dans le document de spécifications qui représentent les pilotes nécessaires à la réponse à la requête. L'ontologie est aussi modifiée de manière à intégrer les concepts de bas niveaux créés par les pilotes. Les acteurs de cette étape sont l'experts du domaine et l'ontologiste.

Étape 3 : Créer les étapes intermédiaires

À cette étape, on crée des requêtes intermédiaires qui servent à contrôler le processus de traduction de l'information. Cela signifie qu'on évalue les niveaux de connaissance intermédiaires qui nous permettent de passer du niveau informationnel de base à celui de notre spécification du système expert. Prenons par exemple la question de savoir si le "cookie" de l'administrateur d'un site web peut avoir été volé en passant par une vulnérabilité XSS. Cette dernière question fait office de spécification puisqu'elle indique à quoi le système doit pouvoir répondre. Comme étapes intermédiaires à cette question, on doit tout d'abord savoir si des pages du site contiennent une vulnérabilité XSS, puis si l'administrateur s'est connecté sur une de ces pages. Cette dernière étape peut elle-même se diviser en deux étapes, soit de savoir quels utilisateurs ont fait une requête sur les pages concernées et lesquels de ces utilisateurs sont administrateurs du site web. Pour savoir si un utilisateur a fait une requête à une page quelconque d'un site web, il faut ensuite premièrement savoir quelles machines ont fait ces requêtes, puis quels utilisateurs étaient connectés sur la machine au moment des requêtes. Cette division en étapes intermédiaires peut s'étirer autant que nécessaire jusqu'à ce qu'on arrive à interroger directement les données brutes. Dans notre exemple, ce sont les événements enregistrés par les systèmes d'exploitation, les serveurs web et les "scanners" de vulnérabilités, ainsi que des données environnementales, telles que les droits des utilisateurs,

les machines et leurs services, etc. Il est possible de ne rien obtenir à cette étape. Cela pourrait indiquer deux choses : soit que le senseur génère une information à un très haut niveau d'abstraction, c'est-à-dire très prêt du langage de l'utilisateur, soit que la question de spécification du système expert est très proche des données. Bien que cela soit possible, ce n'est pas nécessairement souhaitable. En effet, si aucun raisonnement n'est nécessaire sur les données brutes, alors le système expert comme décrit ici perd son intérêt. La plupart du temps, il sera soit possible de diviser une question abstraite en plusieurs questions plus concrètes, soit il y aura un besoin de fusion de deux types d'information. Les requêtes sont écrites en SPARQL et se basent encore une fois sur l'intuition de modélisation de l'expert.

Le résultat de cette étape consiste en de nouvelles entrées dans le diagramme de traduction et dans le document de spécifications qui correspondent aux requêtes intermédiaires créées. L'ontologie devra aussi être ajustée pour prendre en considération les nouveaux concepts et propriétés introduits dans les requêtes. Ses acteurs sont l'expert du domaine et l'ontologiste.

Étape 4 : Expliciter les règles de traduction

Les questions développées à l'étape 3 nous donnent un cadre pour expliciter le travail des agents raisonneurs du système expert. La quatrième étape consiste à trouver la manière la plus efficace pour faire la traduction du niveau d'abstraction de l'information fournie par les senseurs. Cette traduction peut être effectuée de plusieurs manières, soit en logique descriptive, par des règles logiques écrites en SWRL ou par l'application d'algorithmes quelconques.

Nous donnons maintenant un exemple de chaque cas. Pour l'exemple d'une traduction en logique descriptive, prenons une requête qui consiste à obtenir tous les accès à une ressource distante et sa requête intermédiaire qui consiste à obtenir tous les événements "Windows" ayant 5140 comme valeur d'identification. Le traducteur devra ici "traduire" les instances de type "WindowsEvent" qui ont 5140 comme valeur à la propriété "hasID" en instances de type "RemoteFileAccess". Dans ce cas-ci, on ajoutera un axiome d'équivalence à la classe "RemoteFileAccess" à la classe anonyme "WindowsEvent and hasID value 5140" qui spécifie que toutes les instances de type "WindowsEvent" ayant 5140 pour valeur de la propriété "hasID" sont de type "RemoteFileAccess". Comme exemple de traduction SWRL, prenons le cas où un senseur crée des requêtes ayant pour destination une adresse IP ainsi qu'un port et où l'on veut relier ces requêtes aux instances de services concernées avec la propriété "hasDestination". On présente cette requête à l'énoncé 3.2. Finalement, comme exemple de traduction algorithmique, nous pouvons donner le cas des algorithmes qui consistent à récupérer toutes les requêtes réseau de l'ontologie de manière à vérifier si toutes les adresses IP externes concernées sont associées à une machine dans l'ontologie. Lorsqu'une telle ma-

chine n'existe pas, l'algorithme la crée. Cela permet entre autres de créer les machines des attaquants.

$$\texttt{NetworkRequest(?request)} \wedge \texttt{hasDestinationPort(?request, ?port)} \wedge \texttt{hasDestinationIPAddress(?request, ?dstIP)} \wedge \texttt{hasIPAddress(?machine, ?dstIP)} \wedge \texttt{offers(?machine, ?service)} \wedge \texttt{hasPort(?service, ?port)} \rightarrow \texttt{hasDestination(?request, ?service)} \tag{3.2}$$

Étape 5 : Enrichissement de l'ontologie

La dernière étape consiste à enrichir l'ontologie après avoir suivi les 4 étapes de son développement. Ces étapes consistent en fait à créer un système de réponse à des questions en utilisant l'ontologie comme support pour la représentation de l'information. Pour cette raison, l'ontologie développée a une portée très pratique et peut sembler incomplète du point de vue logique ou pour des buts de réutilisabilité, par exemple dans le contexte du web sémantique. On introduit donc une dernière étape dans la méthode qui consiste à l'enrichir avec divers axiomes logiques, de manière à ce qu'elle soit plus complète.

On présente ici une liste non exhaustive des enrichissements possibles :

- Annotations des objets de l'ontologie avec au minimum "rdfs :comment" et "rdfs :label".
- Définition de classes par des axiomes "EquivalentTo" et "SubClassOf" plus complètes.
- Définition d'axiomes "DisjointWith" pour les classes disjointes.
- Caractérisation des propriétés avec les attributs de transitivité, symétrie, etc.
- Description des propriétés avec leur "rdfs :domain" et "rdfs :range".
- Différenciation des instances créées avec la propriété "owl :distinctMembers".

Ces enrichissements permettent une meilleure compréhension de l'ontologie par ses lecteurs et peuvent permettre des résultats intéressants pour l'exploration de l'ontologie utilisée en production. En effet, des enrichissements logiques pertinents peuvent potentiellement faire générer par un raisonneur des inférences intéressantes pour l'utilisateur du système connecté sur son environnement. Cependant, pour être menée à bien, cette étape nécessite un système solide de tests pour s'assurer que les enrichissements de l'ontologie ne brisent pas le comportement initialement prévu lors de son développement.

En plus d'améliorer la compréhensibilité de l'ontologie, l'étape d'enrichissement est aussi

utile pour détecter des erreurs de modélisation, simplifiant ainsi la maintenance du produit. Lors de nos développements, nous avons constaté certaines erreurs entravant l'extensibilité de l'ontologie. Par exemple, lorsque nous avons essayé de modifier le concept d'authentification, nous avons remarqué que notre définition de ce concept était trop simple, parce que nous l'avions introduit dans un contexte où les instances étaient directement créées par un pilote qui devait seulement s'assurer de connecter l'instance à d'autres instances de machines et d'utilisateurs. Pour la réutilisation facile de ce concept, il aurait fallu approfondir un peu plus le concept et introduire d'autres concepts connexes, comme ceux "d'accesseur", "d'autorité" et de "ressource accédée". On voit donc ici que notre modélisation a été incomplète à cause de la portée très pratique de notre processus. L'étape d'enrichissement permet de faire un pas en arrière et d'analyser l'ontologie dans son ensemble.

Un danger de portée ("scope") guette cependant l'ontologiste dans cette étape. En effet, il faut s'assurer que le pas en arrière ne soit pas trop grand. Les mêmes embûches qui avaient été réglées par le pragmatisme de l'effort de traduction visant des réponses spécifiques à des questions spécifiques risquent de refaire surface, surtout en ce qui concerne les définitions suffisantes et nécessaires des classes. Par exemple, dans notre contexte, il était nécessaire et suffisant pour une "Authentification hors ligne" d'être un "Événement Windows", d'avoir 4624 comme valeur d'ID et 2 comme valeur de "logon type". Cela peut sembler étrange. Cependant, nous avons créé cet axiome parce que, dans notre contexte, les seules authentifications hors ligne possibles sont sur des ordinateurs Windows. Donc, dans la phase d'enrichissement, il est justifié de redéfinir l'authentification pour que ce concept soit portable dans un autre contexte, par exemple pour un contexte où les authentifications hors ligne sont possibles sur des machines Linux. Dans ce cas-ci, le pas en arrière est pertinent. Un exemple de pas en arrière trop prononcé que nous avons expérimenté concerne le concept de "Machine". En effet, il a été avancé que nous ne pouvions pas vraiment définir comme propriété nécessaire et suffisante à ce concept le fait d'avoir une adresse IP, parce que celui-ci est beaucoup plus complexe. On pourrait argumenter qu'une machine définit plutôt les choses qui produisent une action avec un mécanisme automatique quelconque. Par exemple, un train est une machine et il n'a pas d'adresse IP. Ce pas en arrière, qui consiste à généraliser le concept de "machine" pour refléter de manière plus exacte la sémantique du terme dans la langue française, est cependant non pertinent puisqu'initialement, nous avons introduit ce terme pour englober les "routeurs", "switchs", etc. en plus des ordinateurs. Le terme aurait pu être simplement "Ordinateur". C'est ce genre de piège qui appelle l'ontologiste à la prudence, car ils peuvent entraîner des coûts élevés en termes d'efforts et de temps.

3.4.2 Exemple concret d'utilisation de ATOM

Nous démontrons maintenant l'utilisation de ATOM en traitant le cas de la détection d'une attaque "heartbleed" réussie de la machine d'un attaquant vers la machine d'une victime.

Étape 0 : Définir les requêtes de l'ontologie en langage naturel

Dans notre exemple, la requête en langage naturel pourrait être formulée de la manière suivante : Est-ce qu'une attaque "heartbleed" a réussi d'une machine A vers une machine B ?

Étape 1 : Traduire la requête en SPARQL

La première étape est de traduire cette définition écrite en langage naturel en SPARQL, basé sur l'intuition de l'expert au sujet de l'encodage souhaitable de cette information dans l'ontologie. Dans notre cas, cette requête a été traduite par l'énoncé SPARQL 3.3 :

```
SELECT ?attack WHERE {
    ?attack rdf:type :HeartbleedAttack .
    ?attack :hasSource ?machineA .
    ?attack :hasDestination ?machineB .
}
```
(3.3)

Cette requête sert de spécification au système expert, c'est-à-dire qu'elle indique la cible d'abstraction visée, ou autrement dit, le travail à effectuer par le système expert. Si nous revenons au concept de droite d'abstraction, nous voyons qu'elle constitue une requête à très haut niveau, c'est-à-dire à un niveau proche du langage naturel. Lors de la conception de cette requête, il faut essayer de réutiliser les concepts et propriétés existants dans l'ontologie. Une fois la requête formulée, il faut ensuite créer les éléments nécessaires dans l'ontologie. Par exemple, dans notre cas, les propriétés "hasSource" et "hasDestination" existaient déjà, mais l'attaque de type "HeatbleedAttack" était nouvelle. Nous avons donc seulement créé ce dernier concept comme une classe à part dans l'ontologie. Lors de ce travail, il faut aussi analyser l'ontologie existante et placer les nouveaux concepts là où c'est pertinent, suivant la taxonomie de l'ontologie. Dans notre exemple, il existait déjà une classe "Attaque". Nous avons donc placé la classe "HeartbleedAttack" comme une sous-classe de celle-ci. Le diagramme à cette étape est composé seulement d'une boîte représentant la requête en langage naturel comme présenté à la Figure 3.5.

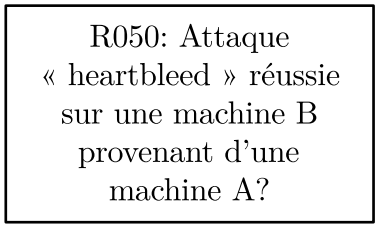


Figure 3.5 Diagramme de la requête haut niveau

On voit ici la notion d'index, "R050" dans notre exemple, qui permet de relier la requête explicitée plus haut et sa composante représentative du diagramme dans le document de spécifications.

Étape 2 : Spécifier les informations brutes

Maintenant que nous savons où nous voulons aller, il faut déterminer les informations brutes disponibles pour répondre à la question, c'est-à-dire qu'il faut déterminer le point de départ informationnel. Encore une fois, cette information est basée sur l'intuition de l'expert qui sait en principe où chercher pour répondre aux questions de spécification du système. Dans notre exemple, il a été déterminé que les senseurs suffisants pour répondre à la question sont un IDS, par exemple "Snort", qui détecte les "charges utiles" de "heartbleed" dans les paquets réseau et un senseur, qu'on a appelé "configurateur", qui a pour fonction de récupérer les informations des machines du réseau surveillé. Les informations nécessaires sont la machine attaquée avec ses caractéristiques, y compris les services présents sur la machine, par exemple, si elle sert un service VPN. On obtient donc provisoirement le diagramme de la Figure 3.6.

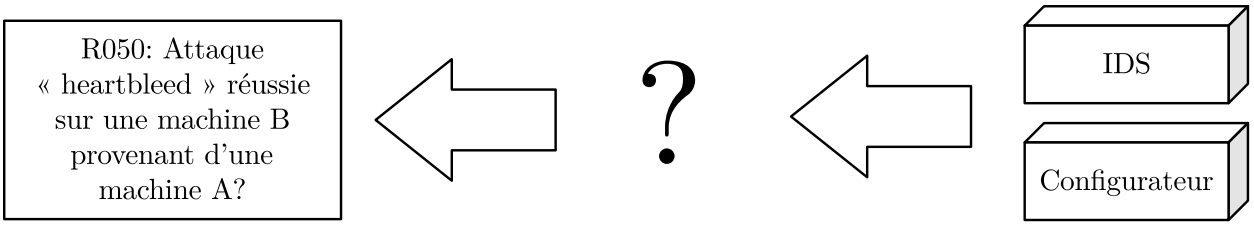


Figure 3.6 Ébauche du flot de traduction

Ce diagramme indique qu'on doit partir de l'information générée par deux senseurs, un IDS et un configurateur, puis exécuter sur cette information une série de traductions indéfinies schématisées par le point d'interrogation de la figure de manière à pouvoir répondre à la question "R050".

Pour permettre une implémentation concrète, une définition plus rigoureuse des senseurs est nécessaire. Pour cette raison, au lieu de décrire les senseurs dans le diagramme, nous utilisons plutôt des identifiants faisant référence à la spécification du pilote qui extrait l'information du senseur. On obtient donc le diagramme illustré à la Figure 3.7.

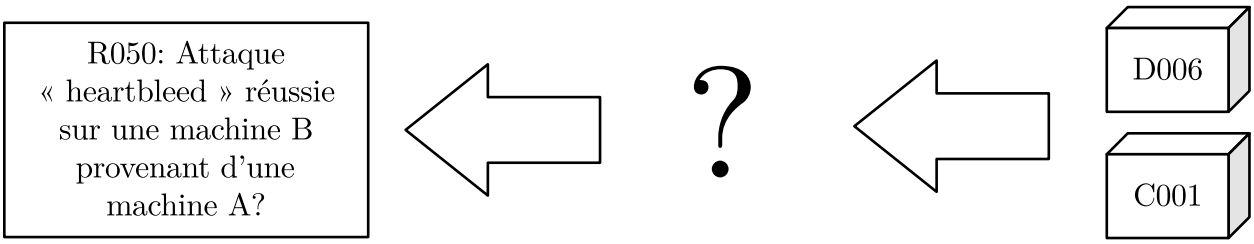


Figure 3.7 Diagramme de flot avec les identifiants des senseurs

À ce diagramme sont associées des entrées dans le document de spécification, comme illustré dans le Tableau 3.2.

Étant donné que chaque senseur peut générer plusieurs instances avec des types et des propriétés différents, il est important de documenter plus en détail de quelle manière l'information doit être générée dans l'ontologie pour chaque cas d'utilisation spécifique. Pour cette raison, en plus des concepts et propriétés, on ajoute dans l'ontologie des exemples typiques d'instances devant être générées par les senseurs documentés et on spécifie le nom de ces instances dans le diagramme. Dans notre exemple, quatre instances d'information brute sont nécessaires pour répondre à la requête "R050". Ces instances sont l'instance de la machine attaquée avec une instance de son service attaqué, une instance de la machine de l'attaquant et une instance représentant le paquet ayant la "charge utile" "heartbleed". Pour l'instance de la machine attaquée et de son service, nous avons créé une instance de type "Machine" qu'on a nommée "sc5_server" ainsi qu'une instance de type "VPNService" qu'on a nommé "sc5_vpnService". Pour l'instance générée par le pilote de "Snort", nous en avons créé une de type "HeartbleedPayload" qu'on a nommé "sc5_heartbleedPacket". En ce qui concerne la dernière instance de la machine de l'attaquant, il n'est pas possible d'obtenir cette information directement d'un senseur. En effet, le senseur a pour rôle unique de traduire l'information extérieure en une représentation interne compréhensible. Dans ce cas-ci, pour entrer dans cette classification, un senseur devrait répertorier toutes les machines connectées sur l'internet pour avoir une modélisation de la machine de l'attaquant, ce qui n'est pas réaliste. Cela étant, il faut raisonner sur l'information interne, c'est-à-dire sur le fait que la machine source du paquet réseau n'existe pas dans le réseau local, afin de créer la machine de l'attaquant. Pour cette raison, la création de cette instance arrive plus tard dans le processus.

Tableau 3.2 Exemple d'entrées dans le document de spécification

ID	Spécification
C001	Crée les instances de la classe `:Machine` en fonction des machines de l'environnement surveillé par le système expert. Les propriétés des instances, si applicables, sont : — `:hasHostname` — `:hasPartition` (Type `:Partition`) — `:hasIPAddress` (e.g. "192.168.3.1") — `:offers` (Type `:Service`) — `:hasOperatingSystem`(Type `:OperatingSystem`, e.g. `:windowsXP`, `:ubuntu`, etc.) Ce pilote doit aussi créer les instances de services offerts par chaque machine modélisée par la propriété `:offers`. Les services ont minimalement les propriétés : — `:hasProtocol` (Type `:Protocol`) — `:hasPort` (e.g. 80) Ils sont instanciés selon leur type suivant les sous-classes de `:Service` disponibles, par exemple `:Website`, `:VPNService`, etc.
D006	Lit les fichiers journaux de "Snort" et crée des instances des classes `:WebScan`, `:SQLInjectionPayload`, `:SensitiveDataAlert`, `:PortScan` et `:HeartbleedPayload`, selon le cas, avec les propriétés suivantes si applicables : — `:hasDomainName` (e.g. "www.site.lan") — `:hasPath` (e.g. "/index") — `:hasDestinationPort` (e.g 80) — `:hasSourceIPAddress` — `:hasDestinationIPAddress`

Nous présentons dans le Tableau 3.3 l'information encodée dans l'ontologie jusqu'à présent.

Ici, nous avons tout de suite discuté de la manière de modéliser différentes informations en se basant sur des intuitions reliées au domaine. Par exemple, l'expert en sécurité savait que pour déterminer si une machine avait été la victime d'une attaque "Heartbleed", il faut savoir si cette machine offre un service vulnérable à cette attaque. L'exemple utilisé pour expliquer ce fait est le cas d'un service VPN ayant un module "openSSL" particulier vulnérable à "Heartbleed". Nous avons donc créé à ce moment une classe "Module" et nous avons instancié différentes instances de cette classe, dont l'instance "openSSL1.0.1c" reliée à notre instance de service VPN. Cette démarche aurait pu être effectuée plus tard lors du travail de traduction, mais nous avons exploité la connaissance et les prédictions de l'expert en sécurité pour effectuer ce travail à ce moment. Au final, nous obtenons le diagramme

partiel des senseurs de la Figure 3.8.

Tableau 3.3 Information encodée dans l'ontologie

Identifiant	Type	Propriétés
sc5_vpnService	`:VPNService`	`:hasModule :openSSL1.0.1c`
sc5_server	`:Machine`	`:offers :sc5_vpnService` `:hasIPAddress 10.5.1.200`
sc5_heartbleedPacket	`:HeartbleedPayload`	`:hasSourceIPAddress 203.1.2.3` `:hasDestinationIPAddress` `10.5.1.200`

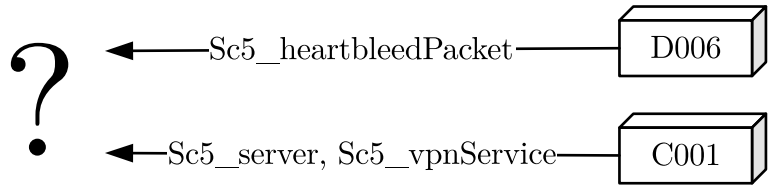


Figure 3.8 Diagramme des senseurs avec instances

Étape 3 : Créer les requêtes intermédiaires

Dans notre exemple, il a été déterminé que la requête "R050" se répond en trois parties. Premièrement, on doit trouver les requêtes possédant la "charge utile" de "heartbleed" et relier celles-ci à leurs acteurs, c'est-à-dire aux machines impliquées dans la requête. Deuxièmement, on doit déterminer les machines qui sont vulnérables à "heartbleed". Finalement, on doit relier ces deux informations de manière à pouvoir générer une attaque "heartbleed" qui doit être instanciée uniquement lorsqu'une requête ayant une "charge utile" "heartbleed" a pour destination une machine ayant la vulnérabilité "heartbleed". De cette manière, on pourra répondre à la requête "R050" qui cherche simplement ce type d'instance. On peut schématiser cette série de questions par le diagramme de la Figure 3.9.

Les requêtes "R047" et "R048" sont créés de la même manière que la requête "R050", c'est-à-dire, en se basant sur l'intuition de l'expert en ce qui concerne la représentation souhaitable de l'information dans l'ontologie. La différence cependant est que lors de la conception de ces requêtes, il faut porter une attention particulière aux senseurs déjà définis ainsi qu'aux instances qui exemplifient le cas. Encore une fois, il faut s'efforcer d'utiliser les concepts déjà existants dans l'ontologie et, lorsqu'un concept est nouveau, il faut l'y ajouter. Dans le tableau 3.4, nous présentons les requêtes SPARQL générées à cette étape ainsi que la requête "R050".

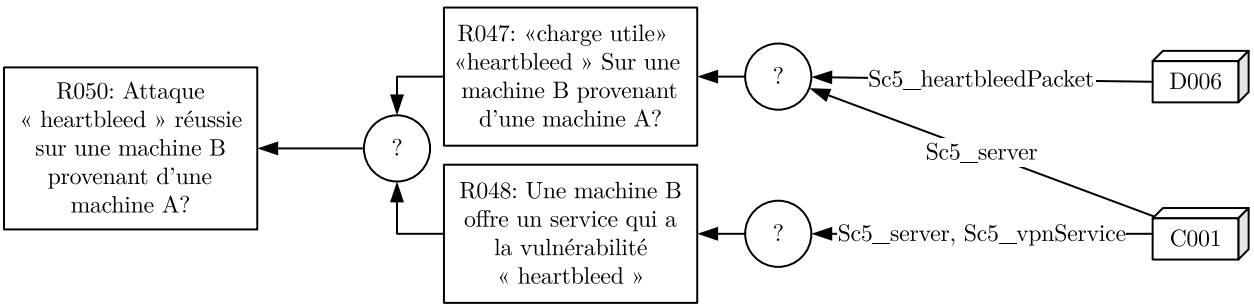


Figure 3.9 Ajout des requêtes intermédiaires

Tableau 3.4 Requêtes SPARQL générées

Identifiant	Requête

R047	`?heartbleedPayload rdf:type :HeartbleedPayload .`
	`?heartbleedPayload :hasSource ?attackerMachine .`
	`?heartbleedPayload :hasDestination ?serverMachine .`
	`?attackerMachine rdf:type :Machine .`
	`?serverMachine rdf:type :Machine .`

R048	`?machine rdf:type :Server .`
	`?machine :offers ?service .`
	`?service :hasVulnerability :heartbleed .`

R050	`?attack rdf:type :HeartbleedAttack .`
	`?attack :hasSource ?machineA .`
	`?attack :hasDestination ?machineB .`

Les cercles ayant pour étiquette des points d'interrogation de la Figure 3.9 illustrent un effort quelconque de traduction à effectuer par le système expert pour lui permettre de répondre à chaque question. Par exemple, pour répondre à la requête "R047", il faut effectuer un certain travail à partir des instances d'information brute "sc5_heartbleedPacket" et "sc5_server" générées par les senseurs "D006" et "C001". La définition de ce travail est faite dans l'étape suivante.

Étape 4 : Expliciter les règles de traduction

On commence par l'effort de traduction à effectuer entre les senseurs et la requête "R047". Lorsqu'on analyse cette requête, on voit qu'il nous manque une donnée pour y répondre, soit la machine de l'attaquant. Donc, un premier algorithme de traduction devra lire toutes les requêtes impliquant des machines et devra créer les instances de machines qui n'existent pas dans l'ontologie, en se basant sur les adresses IP des machines sources et destinations des re-

quêtes. Nous sommes ici conscients qu'utiliser l'adresse IP comme identifiant primaire d'une instance de machine peut ne pas être viable pour déterminer si elle est présente ou pas dans l'ontologie. Nous conservons tout de même cette notion par souci de simplicité et parce que cela démontre bien notre cas. Il est entendu qu'une autre méthode d'identification plus précise pourrait être proposée à ce niveau, ou au minimum, un mécanisme de détermination d'équivalence d'instances de machines qui se baserait sur un ensemble de caractéristiques, comme par exemple, l'adresse MAC de leur carte réseau. On introduit donc un raisonneur "T014" qui a la fonction de créer ces instances de machines externes et on crée une instance pour la machine de l'attaquant de notre exemple dans l'ontologie, "sc5_attackerMachine", ayant pour propriété l'adresse IP de la source de l'instance "sc5_heartbleedPacket". On a maintenant toutes les données nécessaires pour répondre à la requête. Cependant, on remarque que la requête tient pour acquis que l'instance "sc5_heartbleedPacket" est directement reliée aux instances de type "Machine" qui constituent sa source et sa destination. Ce n'est actuellement pas le cas. On peut cependant inférer ces liens en se basant sur les adresses IP des machines et sur les adresses IP source et destination spécifiées dans la requête. On introduit donc deux nouvelles règles, "T012" et "T013", qui ont pour but respectivement de relier directement les requêtes à leur machine destination et à leur machine source. Ces règles sont spécifiées dans le Tableau 3.5 et on obtient alors le diagramme illustré à la Figure 3.10 pour cette séquence de traduction.

Tableau 3.5 Exemple de règles de traduction

ID	Type	Description
T012	SWRL	`Event(?event)` ∧ `hasDestinationIPAddress(?event, ?x)` ∧ `hasIPAddress(?machine, ?x)` → `hasDestination(?event, ?machine)`
T013	SWRL	`Event(?event)` ∧ `hasSourceIPAddress(?event, ?x)` ∧ `hasIPAddress(?machine, ?x)` → `hasSource(?event, ?machine)`

T014	Algorithme	Récupère toutes les instances de type `:Event` qui ont les propriétés `:hasDestinationIPAddress X` ou `:hasSourceIPAddress X`.
		S'il n'existe pas de machine ayant l'adresse IP X dans le système, alors il faut la créer :
		`?newMachine rdf:type :ExternalMachine`
		`?newMachine :hasIPAddress X`
		Où `:ExternalMachine` est sous-classe de `:Machine`

On peut maintenant s'attaquer à la requête "R048". En analysant la requête SPARQL, on

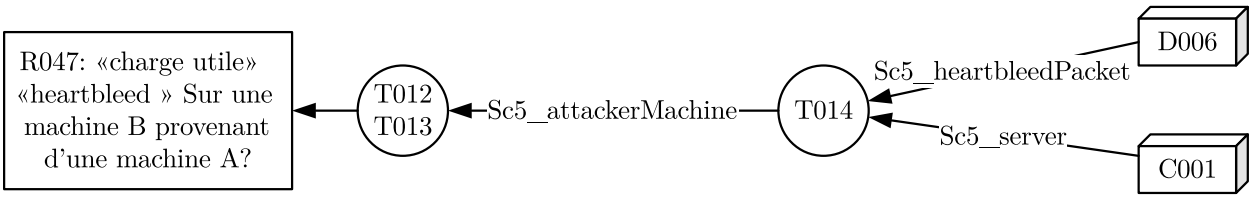


Figure 3.10 Diagramme de la traduction de R047

constate qu'il nous manque la notion de "Serveur" et qu'on doit attacher la vulnérabilité "heartbleed" aux services qui l'ont avec la propriété "hasVulnerability". On peut noter ici qu'il y aurait plusieurs manières de modéliser l'idée exprimée par la requête "R048". Par exemple, est-il vraiment nécessaire de faire référence à la machine attaquée comme étant un serveur ? On pourrait utiliser plus simplement le concept de "Machine" déjà existant dans l'ontologie. On peut aussi se questionner sur la manière d'indiquer la vulnérabilité. Ici, nous avons opté pour l'instanciation de la vulnérabilité et le rattachement de cette instance aux services qui l'ont. Il y aurait eu plusieurs autres manières de le faire. Par exemple, nous aurions pu créer une classe des machines vulnérables à "heartbleed", par exemple "HeartbleedVulnerableMachine", et trouver une manière de faire cette classification automatiquement. Dans ce cas-ci, la requête SPARQL aurait été simplement l'énoncé 3.4.

```
?machine rdf:type :HeartbleedVulnerableMachine .
```
(3.4)

Nous aurions aussi pu créer une classe de services ou de modules vulnérables et interroger l'ontologie à ce niveau. En gardant l'idée de l'instance de vulnérabilité, il n'est pas évident de faire le choix d'attacher l'instance aux services plutôt qu'aux modules vulnérables ou aux machines. On constate donc la nécessité d'un critère de modélisation. Celui-ci peut difficilement être le rasoir d'Occam dans ce contexte. En effet, la requête présentée ci-dessus est plus simple que la requête "R048". Cependant, on a l'intuition qu'il n'est pas souhaitable d'avoir une classe "HeartbleedVulnerableMachine". En effet, on sait que la vulnérabilité est une entité séparée des machines. Rien ne nous empêche de créer la classe des machines vulnérables à "heartbleed", mais notre intuition nous dit qu'il faudrait rattacher une instance de la vulnérabilité "heartbleed" quelque part dans la modélisation. On voit donc que nos critères de modélisation sont plutôt l'intuition de l'expert du domaine et une tentative de se rapprocher du langage naturel. En effet, on dira "Le service X a la vulnérabilité Y." plutôt que "Le service X est un service de type Z.", où "Z" est la classe des services ayant la vulnérabilité "Y". C'est pour exploiter cette idée que nous faisons tout d'abord la conception

de nos requêtes et qu'on essaie de calquer la modélisation sur ces requêtes. L'idée est de se rapprocher le plus possible de la représentation mentale de l'expert en sécurité pour que la modélisation du système soit la plus intuitive possible. Ainsi, étant donné que l'expert a pour premier réflexe de s'enquérir d'une sous-classe des machines, c'est-à-dire des serveurs, on modélise cette notion dans l'ontologie.

Une autre notion importante pour la modélisation est la simplification de l'effort de traduction. Ainsi, une traduction par logique descriptive est préférable à une traduction par des règles SWRL qui est elle-même préférable à une traduction algorithmique. En premier lieu, on essaie donc toujours d'encoder la traduction en logique descriptive. Dans le cas présent, on peut définir la classe "Server" comme étant la classe des "Machine" qui offrent un service, c'est-à-dire que dès qu'une machine "offers" une instance de type "Service", on peut considérer que c'est un serveur. Ainsi, à l'exécution d'un raisonneur de logique descriptive, notre instance "sc5_server" de type "Machine" sera automatiquement classifiée comme étant une instance de type "Server". Dans notre diagramme, nous encodons ce principe par l'identifiant "T041".

La dernière étape de ce travail de traduction est d'attacher la vulnérabilité "heartbleed" préexistante dans notre ontologie aux services qui l'ont. À ce moment, on doit se poser la question au sujet de ce qui rend un service vulnérable. La réponse de l'expert en sécurité était qu'un service est toujours vulnérable à cause d'une librairie ou d'une composante quelconque qui permet cette vulnérabilité. Nous avons donc décidé d'englober tous les cas possibles en introduisant le concept de "Module" comme décrit dans la section des senseurs. Le concept de classe d'objets vulnérables a du sens ici puisqu'on peut prédéterminer l'ensemble des modules qui provoquent la vulnérabilité dans un service qui l'implémente. Pour cette raison, nous avons créé une classe "HeartbleedVulnerableModule" définie comme une énumération d'instances de module, dans notre cas, "openSSL1.0.1a", "openSSL1.0.1b", etc. Contrairement au cas des services qui doivent être générés dynamiquement en fonction de l'environnement, les modules sont des entités fixes dans notre système et il était donc beaucoup plus logique de créer une classe "HeartbleedVulnerableModule" plutôt qu'une classe "HeartbleedVulnerableService". La même justification tient pour la classe "HeartbleedVulnerableMachine". La règle "T043" est utilisée pour la classification des instances de module vulnérables. Comme on sait que le senseur défini à l'étape 2 doit relier les services des machines créées à leurs modules lorsque pertinent, il ne reste qu'à relier les services à la vulnérabilité "heartbleed" lorsqu'ils ont un module de type "HeartbleedVulnerableModule". Il n'y a pas de manière évidente de faire cela en logique descriptive et on utilise donc une règle SWRL qu'on a étiquetée par

l'identifiant "T044" et qui est présentée dans l'énoncé 3.5.

$$\begin{aligned} &\texttt{Service(?service)} \wedge \texttt{hasModule(?service, ?module)} \wedge \\ &\quad \texttt{HeartbleedVulnerableModule(?module)} \rightarrow \\ &\texttt{hasVulnerability(?service, heartbleed)} \end{aligned} \tag{3.5}$$

On remarque ici que nous avons fait référence dans la règle à la classe "Service". En effet, peu importe le type de service, s'il a un module vulnérable à "heartbleed", il aura cette vulnérabilité. La généralisation a donc du sens ici. Le problème est que le senseur "C001" a créé une instance de type "VPNService" et non pas de type "Service". Donc, sans traitement préalable, cette instance ne serait pas considérée par la règle "T044". C'est ici que les relations taxonomiques trouvent leur utilité. En effet, l'ontologie définit la classe "VPNService" comme étant une sous-classe de la classe plus générale "Service". Ainsi, lorsqu'un raisonneur fera son travail d'inférence, il déterminera que toutes les instances de type "VPNService" sont aussi de type "Service". Nous avons schématisé ce travail du raisonneur par la règle "T059".

Bien qu'il soit préférable dans notre cas d'utiliser les règles de logique descriptive plutôt que les règles SWRL pour l'inférence, il peut arriver que même si on a la possibilité d'utiliser la logique descriptive, on décide tout de même d'utiliser SWRL. En effet, il est possible qu'une traduction en logique descriptive introduise de nouveaux concepts, augmentant ainsi la complexité de l'ontologie, ce qui n'est pas souhaitable. Par exemple, dans le cas de la règle "T044", nous aurions pu créer une classe "HeartbleedVulnerableService" définie comme équivalente à la classe des instances qui sont de type "Service" et qui sont en relation avec une instance de type "HeartbleedVulnerableModule" par la relation "hasModule". De cette manière, le service de notre exemple serait automatiquement classé comme étant de type "HeartbleedVulnerableService". Il ne resterait alors qu'à définir cette classe comme sous-classe de la classe ayant une relation "hasVulnerability" avec l'instance "heartbleed". Dans ce cas-ci, comme nous l'avons expliqué plus haut, nous ne souhaitons pas l'introduction de la classe "HeartbleedVulnerableService" et nous avons donc plutôt opté pour une modélisation par règle SWRL.

On peut maintenant répondre à la requête "R048" et on obtient le diagramme de la Figure 3.11. Le Tableau 3.6 illustre le flot de traduction qui permet de répondre à la requête de manière abrégée.

La dernière étape de notre démarche est de déterminer le traitement pour répondre à la requête "R050" en nous basant sur les résultats de modélisation obtenus par les requêtes "R047" et "R048". Ce traitement est représenté par le point d'interrogation de la Figure 3.12.

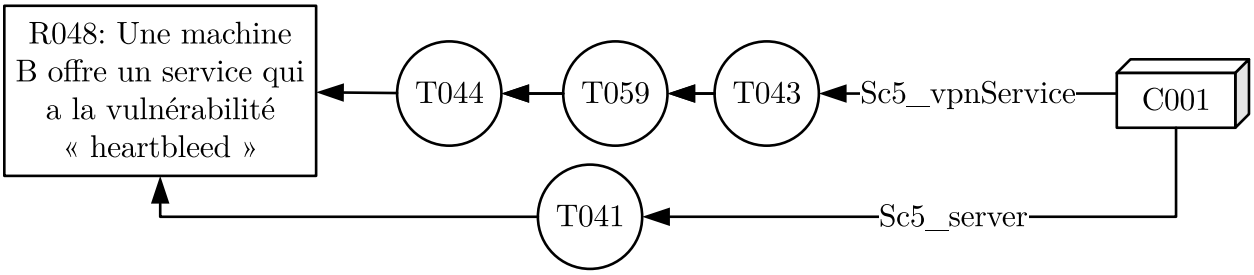


Figure 3.11 Diagramme de la requête R048

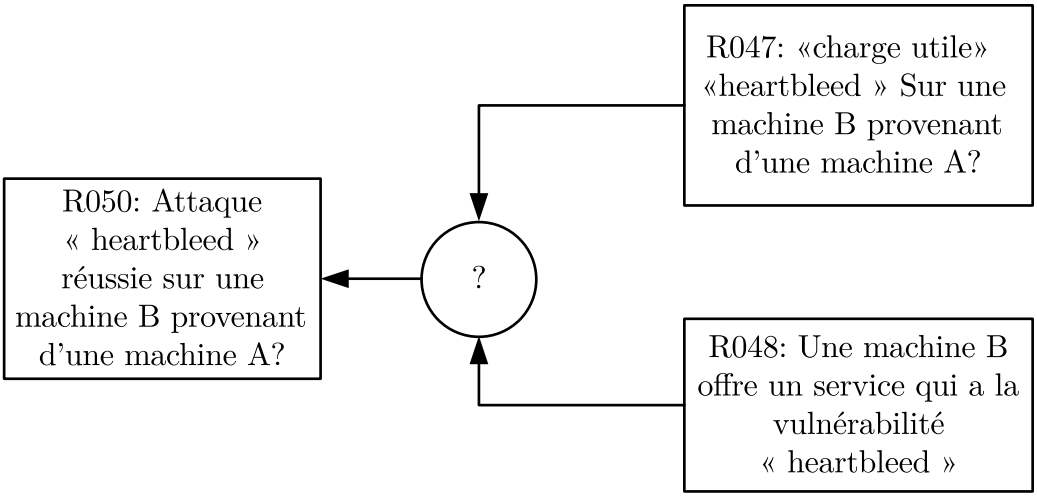


Figure 3.12 Diagramme partiel R050

En regardant la requête “R050”, on voit qu'il faut créer une instance de type “HeartbleedAttack” en se basant sur l'information disponible. Cette information est composée d'un événement de type “HeartbleedPayload” relié à ses machines source et destination ainsi que la machine destination étant vulnérable à “heartbleed”. De manière à simplifier la récupération des machines vulnérables à “heartbleed”, on introduit une règle SWRL qui, pour toute machine ayant un service vulnérable, attache la vulnérabilité du service à cette machine, et ce, pour tous ses services. Cette règle, étiquetée “T045”, est présentée dans le Tableau 3.7. On peut finalement introduire la règle “T057” qui récupère les événements de type “HeartbleedPayload” avec leurs machines source et destination pour lesquels la machine destination a la vulnérabilité “heartbleed” et qui crée, pour chaque résultat obtenu, une instance d'attaque “heartbleed” reliée à ces machines, comme présenté dans le Tableau 3.7. Le diagramme de la Figure 3.13 présente le flot de traduction complet de la requête R050.

Pour récapituler, ce que nous obtenons au final est une méthode propre à l'implémentation

Tableau 3.6 Flot de traduction de la requête R048

Étape	Résultat

Exécution du senseur C001	Création de l'instance de service :
	`sc5_vpnService rdf :type :VPNService`
	`sc5_vpnService :hasModule :openSSL1.0.1c`
	Création de l'instance de serveur :
	`sc5_server rdf:type :Machine`
	`sc5_server :offers sc5_vpnService`

Application de la règle T041	Classification des machines qui offrent un service comme serveurs : `sc5_server rdf:type :Server`
Application de la règle T043	Par définition d'énumération de `:HeartbleedVulnerableModule` : `:openSSL1.0.1c rdf:type :HeartbleedVulnerableModule`
Application de la règle T059	Puisque la classe `:VPNService` est une sous-classe de `:Service`, toutes les instances de type `:VPNService` sont aussi de type `:Service` - `sc5_vpnService rdf:type :Service`
Application de la règle T044	Toutes les instances de type :Service qui ont un module vulnérable à "heartbleed" sont mis en relation avec l'instance de vulnérabilité "heartbleed" : `sc5_vpnService :hasVulnerability heartbleed`

Requête R048	On peut maintenant remplacer `?machine` par `sc5_server` et `?service` par `sc5_vpnService` :
	`(?machine = sc5_server) rdf:type :Server`
	`(?machine = sc5_server) :offers (?service = sc5_vpnService)`
	`(?service = sc5_vpnService) :hasVulnerability heartbleed`

qui permet de passer d'une information à très bas niveau générée par des senseurs, soit des machines et des requêtes réseau, à une information à très haut niveau, soit une attaque.

Tableau 3.7 Règles pour R050

ID	Type	Description
T045	SWRL	`Machine(?machine)` $\wedge$ `offers(?machine, ?service)` $\wedge$ `hasVulnerability(?service, ?vulnerability)` $\rightarrow$ `hasVulnerability(?machine, ?vulnerability)`

T057	Algorithme	Un algorithme qui exécute la requête SPARQL :
		`SELECT ?attackerMachine, ?serverMachine WHERE {`
		`?event rdf:type :HeartbleedPayload .`
		`?event :hasSource ?attackerMachine .`
		`?event :hasDestination ?serverMachine .`
		`?serverMachine :hasVulnerability :heartbleed .`
		`}`
		et qui crée pour chaque résultat une instance de type `:HeartbleedAttack` avec les propriétés :
		`:hasSource ?attackerMachine`
		`:hasDestination ?serverMachine`
		`:isComposedOf ?event`

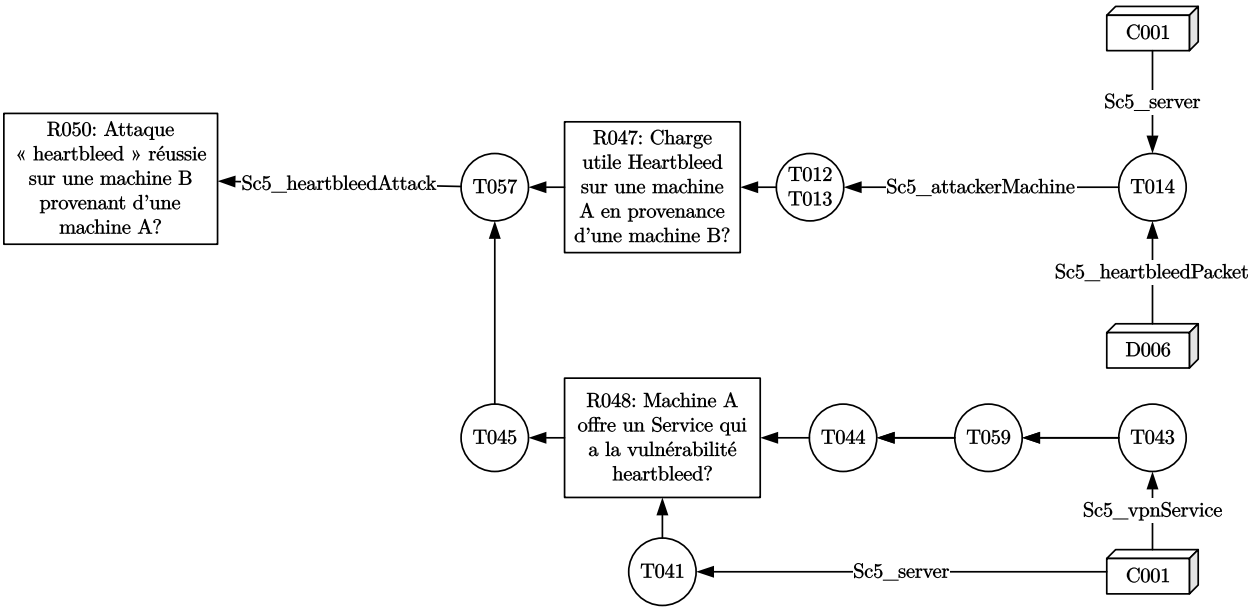


Figure 3.13 Diagramme du flot de traduction complet de la requête R050

Développement de l'ontologie

Au fil du processus, nous avons introduit plusieurs éléments dans l'ontologie que nous présentons ici. Nous présentons les concepts dans l'ordre où ils ont été introduits.

Pour la première étape, la classe "HeartbleedAttack" a été créée comme sous-classe de la classe préexistante "Attack". Une mention est aussi faite par la requête de l'étape 1 aux propriétés "hasDestination" et "hasSource".

Dans la deuxième étape, les classes "VPNService" et "Machine" ont été introduites à cause du senseur "C001". Pour relier les deux classes, la propriété "offers" a été créée. Pour caractériser les machines, nous avons ensuite ajouté une propriété "hasIPAddress". Étant donné que nous voulons nous adresser aux vulnérabilités de modules, nous avons aussi créé une classe "Module" et une propriété "hasModule" qui permet de relier les services à leurs modules. C'est encore "C001" qui s'occupe de faire cette relation. En ce qui concerne le pilote "D006" qui s'interface avec "Snort", nous avons créé une classe "HeartbleedPayload" et des propriétés "hasSourceIPAddress" et "hasDestinationIPAddress" documentant respectivement la source et la destination du "payload".

La troisième étape de la méthode a introduit la notion de "Server" par la requête "R048", mais on ne l'a pas définie plus en détail. C'est aussi dans cette étape que la propriété "hasVulnerability" a été créée, encore une fois à cause de la requête "R048". Comme objet de cette propriété, on a créé une instance de type "owl :Thing", "heartbleed" pour instancier la vulnérabilité.

Dans la quatrième étape, nous avons introduit plusieurs classes. Tout d'abord, afin de définir un serveur comme étant une machine qui offre des services, nous avons créé la classe "Service" sous laquelle on a placé la classe de service créée à l'étape 2, "VPNService". Nous avons ainsi pu enrichir la classe "Server" par une condition nécessaire et suffisante. Ensuite, pour des fins de généralisation de la règle de traduction "T012", nous avons introduit la classe "Event" comme super classe de "HeartbleedPayload". Puis, nous avons créé la classe "HeartbleedVulnerableModule" comme une classe statique où nous avons instancié les modules vulnérables concernés, par exemple "openSSL1.0.1c". Finalement, nous avons ajouté la classe "ExternalMachine" pour les machines externes créées par la règle de traduction "T014". Cette règle a aussi provoqué la création d'une propriété "isComposedOf" pour pouvoir relier les instances d'attaque créées aux événements impliqués dans la détection.

Étape 5 : Enrichissement de l'ontologie

La dernière étape consiste à essayer d'enrichir les concepts introduits dans l'ontologie tout en respectant les intentions initiales, comme nous l'avons expliqué plus tôt.

Comme enrichissement relativement facile, on peut tout d'abord annoter l'ontologie et spécifier les classes disjointes. Nous n'illustrerons pas ici l'annotation des concepts, car ce travail est simple. En ce qui concerne les classes disjointes, on peut simplement énoncer tout d'abord que les classes "Service", "Event", "Module", "Machine" et "Attack" sont disjointes. Étant donné que la classe "VPNService" est sous-classe de "Service", il n'est pas nécessaire de spécifier qu'elle est disjointe des autres classes. Cela vaut aussi pour les classes "HeartbleedPayload", "HeartbleedVulnerableModule" et "HeartbleedAttack". En ce qui concerne les classes "ExternalMachine" et "Server", on ne peut pas dire qu'elles sont disjointes l'une de l'autre, car une machine externe pourrait être un serveur.

Un enrichissement un peu plus complexe, mais tout de même facile, est l'ajout des domaine et portée des propriétés. Par exemple, dans notre cas, on peut affirmer que le domaine de la propriété "isComposedOf" est "Attack" et que sa portée est "Event". Il peut arriver dans ce genre d'enrichissement qu'on ajoute des éléments dans l'ontologie. Par exemple, en ce qui concerne la propriété "hasVulnerability" introduite à l'étape 3, on voudrait que son domaine soit la classe des vulnérabilités. Cependant, nous n'avons pas introduit ce concept à ce moment. On peut donc l'introduire ici et classifier l'instance "heartbleed" comme appartenant à cette nouvelle classe.

Un autre type d'enrichissement est de caractériser les propriétés de l'ontologie. En ce qui concerne les propriétés de données, on peut énoncer si elles sont fonctionnelles. Par exemple, dans notre cas, les propriétés "hasDestinationIPAddress" et "hasSourceIPAddress" sont fonctionnelles. Comme présenté dans la section 2.1.3, les propriétés d'objet ont plusieurs caractéristiques possibles. Elles peuvent être fonctionnelles, fonctionnelles inverse, transitives, symétriques, asymétriques, réflexives et non réflexives. On peut donc caractériser les propriétés introduites suivant la compréhension qu'on en a avec ces caractéristiques. Prenons pour exemple, la propriété "offers". Elle n'est pas fonctionnelle, car une machine peut offrir plus d'un service. Elle est inversement fonctionnelle parce qu'un service peut être offert seulement par une machine. Cela pourrait être autrement, mais dans notre cas, le pilote "C001" crée une instance de service pour chaque machine individuellement. Elle est transitive, car si une machine offre un service qui offre un service, on peut considérer que la machine offre aussi ce deuxième service. Elle n'est pas symétrique parce que bien qu'une machine puisse offrir un service, un service ne peut pas offrir une machine. Étant donné cela, on peut conclure qu'elle est asymétrique. Elle n'est pas réflexive parce qu'une machine ne peut pas s'offrir elle-même

et finalement, étant donné cela, elle est “non réflexive”. Le même genre de raisonnement peut s’appliquer à toutes les propriétés de l’ontologie.

On donne comme dernier exemple d’enrichissement la possibilité de définir les classes par des axiomes logiques, soit nécessaires, soit nécessaires et suffisants. Par exemple, dans notre cas, on peut définir comme axiome nécessaire qu’une machine a une adresse IP en ajoutant l’axiome “`hasIPAddress some xsd:string`”. Cela implique que si un objet est une machine, alors il a nécessairement une adresse IP. Cela peut ne pas être vrai dans un autre contexte, mais, puisque nous étudions une attaque réseau, ce l’est dans le nôtre. On peut alors se demander si le fait d’avoir une adresse IP fait qu’un objet appartient à la classe “Machine”. Dans notre cas, ce l’est et on peut ainsi placer cet axiome comme nécessaire et suffisant. Cette démarche peut être effectuée pour toutes les classes.

CHAPITRE 4 DONNÉES PRODUITES PAR L'APPLICATION DE ATOM

En plus de la solution présentée dans notre chapitre 3, nous avons obtenu des données et artefacts utiles à son évaluation et pour répondre à nos questions de recherche. Nous les présentons dans ce chapitre.

4.1 Artefacts produits par le processus de recherche

Notre processus peut se diviser en deux, soit en la modélisation relative aux environnements informatiques et en celle relative au contrôle aérien. Nous présentons donc les artefacts produits suivant cette division. La méthode de développement que nous proposons a été entièrement suivie en ce qui a trait à la sécurité des environnements informatiques, mais elle ne l'a été que partiellement pour la modélisation du contrôle aérien. Les raisons à cela sont une moins grande disponibilité des experts en contrôle aérien et une plus grande expérience de l'ontologiste en informatique. Nous présentons tout de même nos résultats pour les deux contextes car ils ont tous deux été impliqués dans le développement de la méthode, les ontologies produites sont des résultats de l'application de celle-ci et elles sont utiles pour répondre à nos questions de recherche.

4.1.1 Artefacts de la modélisation des environnements informatiques

En ce qui concerne les environnements informatiques, nous obtenons une description de trois scénarios d'attaque que le système expert doit pouvoir détecter, décomposés en une série de requêtes, et les artefacts générés par ATOM, soit une ontologie, un diagramme de traductions et un document de spécification. Dans ce cas-ci, l'ontologie est complète, c'est-à-dire que l'étape d'enrichissement a été exécutée sur celle-ci. De plus, les autres documents sont dans un état où il serait possible de faire l'implémentation du système décrit. Nous ne présentons ici qu'un survol des scénarios produits, puisqu'ils sont utilisés dans notre évaluation, mais le lecteur est invité à lire Ducharme (2017) pour une description plus détaillée des artefacts produits.

Scénario 1 : Vol d'information via un média amovible malveillant

Le premier scénario concerne le vol d'un fichier de propriété intellectuelle par un attaquant qui utilise comme vecteur d'entrée la macro malveillante d'un fichier "Excel" situé sur une clé USB. Un utilisateur qui a trouvé la clé USB est encouragé par "social engineering" à

exécuter la macro. Une fois dans le réseau, l'attaquant exploite une vulnérabilité pour avoir les privilèges des administrateurs et exfiltre le fichier convoité.

L'environnement dans lequel le premier scénario se déroule est le réseau informatique d'une entreprise géré par le service "Active Directory" et dont les machines connectées au réseau ont pour système d'exploitation une version entreprise de "Windows". Il est assumé que les privilèges des utilisateurs y sont configurés correctement.

Les individus impliqués par le scénario sont :

- Une partition réseau
- Un fichier de propriété intellectuelle sur cette partition protégé de manière à ce que seuls les administrateurs et le personnel approprié y aient accès
- Un groupe "Active Directory" avec une configuration de privilèges appropriée pour les utilisateurs et les administrateurs
- Un administrateur réseau dont le nom d'utilisateur est "Bob"
- Une machine dont l'adresse IP est "10.1.2.3", le nom d'hôte est "WINDOWS-PC" et le système d'exploitation est "Windows 7"
- Un utilisateur propriétaire de cette machine dont le nom d'utilisateur est "Francis"
- La machine externe de l'attaquant dont l'adresse IP est "96.20.1.2"
- Une clé USB contenant un fichier "Excel" possédant une macro malveillante

Le scénario se déroule comme suit :

1. L'utilisateur "Francis" insère la clé USB malveillante dans son ordinateur et exécute la macro du fichier "Excel".
2. La macro exécutée ouvre un canal de communication entre l'ordinateur de l'attaquant et celui de "Francis", ce qui permet à l'attaquant d'exécuter des commandes sur cette machine.
3. L'attaquant utilise la machine infectée pour accéder au fichier de propriété intellectuelle sur la partition réseau. Cependant, étant donné que l'utilisateur "Francis" n'a pas les droits d'accès au fichier, cette tentative est refusée.
4. L'attaquant effectue une exploitation de type "Mimikatz" de manière à récupérer les identifiants de l'utilisateur "Bob" qui se connecte à un moment sur la machine de "Francis".
5. L'attaquant utilise le compte utilisateur de l'administrateur "Bob" pour accéder au contenu du fichier protégé et pour l'exfiltrer sur une machine externe au réseau.

Ce scénario a été décomposé en cinq requêtes de haut-niveau qui ont été elles-mêmes décomposées en 22 requêtes intermédiaires. Son diagramme de traduction implique 8 pilotes de senseurs et 29 règles de traduction diverses.

Scénario 2 : Vol d'information via une vulnérabilité Web

Dans le deuxième scénario, un attaquant exploite la vulnérabilité XSS d'un site web de commerce électronique dans le but de récupérer le "cookie" d'un administrateur. Il peut ainsi se connecter sur la zone d'administration où il exploite une vulnérabilité pour effectuer une injection SQL et ainsi, récupérer les numéros de carte de crédit enregistrés dans la base de données du site web.

L'environnement dans lequel se déroule le deuxième scénario est un serveur web accessible depuis l'internet dont le système d'exploitation est une version de Linux et qui sert des pages web en utilisant les technologies PHP, Apache et MySQL. Le réseau dans lequel se situe ce serveur comporte quelques mécanismes de défense. On y retrouve l'IDS "Snort", le pare-feu web "ModSecurity" et un filtreur web, "SquidGuard".

Les individus impliqués dans le scénario sont :

- Un groupe "ActiveDirectory" dans lequel on retrouve les administrateurs du site web
- Un administrateur dont le nom d'utilisateur est "Vivien" qui est membre de ce groupe
- Une machine externe ayant pour adresse IP "132.207.1.2"
- Une machine externe ayant pour adresse IP "96.1.2.3" de laquelle l'attaquant exécute la plupart de l'attaque
- Le serveur web se faisant attaquer ayant pour adresse IP "10.2.1.100" et qui contient les numéros de carte de crédit dans sa base de données
- L'ordinateur de l'administrateur "Vivien" ayant pour adresse IP "10.2.1.101" et pour nom d'hôte "VIVIEN-PC"
- Un site web ayant pour nom de domaine "site.lan"
- La plateforme de commerce électronique "Wordpress" "WooCommerce" dont le numéro de version est "2.3.2"
- Deux pages web servies par le serveur web dont les chemins d'accès sont "/about" et "/forum"
- Une page web d'authentification possédant "/admin" pour chemin d'accès
- Une page de la section administrative du site web dont le chemin d'accès est "/admin_plugins"

Le scénario se déroule comme suit :

1. L'attaquant exécute un balayage de vulnérabilité web sur le site web en utilisant l'outil "W3af" et y trouve une vulnérabilité XSS.
2. L'attaquant exploite la vulnérabilité XSS sur le forum du site web. Pour ce faire, il place sur une page du forum un code Javascript qui tente de récupérer une image et qui envoie le "cookie" de l'utilisateur au serveur distant. L'administrateur "Vivien"

qui accède à cette page envoie donc à son insu son “cookie” à l’attaquant.

3. L’attaquant utilise le “cookie” de “Vivien” pour naviguer sur le site web en tant qu’administrateur.
4. L’attaquant exploite la vulnérabilité “WPVDB-7846” de la version “2.3.2” de “WooCommerce” qui permet une injection SQL et il récupère de cette manière la table de la base de données qui contient les numéros de carte de crédit des clients.

Ce scénario a été divisé en cinq requêtes de haut-niveau, qui ont elles-mêmes été décomposées en 16 requêtes intermédiaires. Il implique l’utilisation de sept pilotes et de 23 règles logiques. Quatre de ces pilotes et douze de ces règles logiques ont été récupérées du premier scénario.

Scénario 3 : “Rançongiciel” via la vulnérabilité “Heartbleed”

Dans le troisième scénario, un attaquant récupère les identifiants d’un utilisateur du réseau d’une entreprise en exploitant la vulnérabilité “Heartbleed” du serveur VPN de cette entreprise. Cette vulnérabilité permet en effet de récupérer frauduleusement de l’information dans la mémoire d’un ordinateur qui la possède. Une fois connecté au réseau, l’attaquant peut alors se connecter sur une machine de l’entreprise et chiffrer tous ses fichiers en utilisant un “rançongiciel”. Pour les déchiffrer, l’entreprise devra payer l’attaquant pour obtenir la clé de déchiffrement.

L’environnement dans lequel se déroule le troisième scénario est le réseau d’une entreprise géré par “Active Directory”. Dans ce réseau, on retrouve une machine “linux” utilisée comme serveur VPN qui accepte des connexions entrantes provenant de l’extérieur du réseau et diverses machines “Windows”. Un SDI est en place entre le réseau local et le réseau externe.

Les individus impliqués dans le scénario sont :

— Une librairie “OpenSSL” ayant pour version “1.0.1c” qui est vulnérable à “Heartbleed”
— Un service VPN utilisant cette librairie
— Un serveur linux offrant ce service et ayant pour adresse IP “10.5.1.200”
— La machine externe de l’attaquant dont l’adresse IP est “201.1.2.3”
— Une machine interne ayant pour adresse IP “10.5.1.201” et pour nom d’hôte “ALICE-PC”
— Un compte utilisateur compromis par l’attaquant dont le nom d’utilisateur est “alice”
— Un serveur RDP

Le scénario se déroule comme suit :

1. L’attaquant utilise l’outil “nmap” pour effectuer un balayage de port sur une plage d’adresses IP de manière à trouver des machines vulnéables à “Heartbleed”. Le serveur

VPN de l'entreprise est détecté dans ce balayage.

2. L'attaquant exploite la vulnérabilité "Heartbleed". Pour ce faire, il envoie des paquets "heartbeat" malveillants au serveur VPN. Ce dernier répond par une partie de la mémoire qui ne devrait pas être accédée de l'extérieur. L'attaquant envoie des paquets malveillants jusqu'à ce qu'il reçoive les identifiants de connexion de l'utilisateur "alice".
3. L'attaquant utilise ensuite les identifiants de "alice" pour se connecter au réseau de l'entreprise avec sa machine.
4. L'attaquant utilise l'outil d'analyse de trafic réseau "Wireshark" pour faire la reconnaissance du réseau de l'entreprise. Il obtient ainsi l'adresse du serveur RDP de l'entreprise. Il s'y connecte en utilisant les identifiants de Alice, ce qui lui donne un accès graphique à un ordinateur du réseau de l'entreprise.
5. À partir de cette machine, l'attaquant télécharge un "rançongiciel" qui chiffre tous les fichiers de l'ordinateur à l'exception des fichiers nécessaires à son fonctionnement.

Ce scénario a été divisé en cinq requêtes de haut-niveau. Il a impliqué l'utilisation de sept pilotes, dont quatre réutilisés et de 21 règles de traduction, dont sept récupérées des scénarios précédents.

L'ontologie

En ce qui concerne l'ontologie produite, elle comporte 68 classes, 51 propriétés et 93 individus, pour un total de 792 axiomes. Les classes de l'ontologie se divisent principalement en trois catégories. Nous retrouvons tout d'abord les classes d'environnement, telles que "File", "OperatingSystem", "User", etc. Puis, nous avons les classes d'événements qui permettent de modéliser les authentifications ("Authentication"), les partitions montées ("MountedPartition"), les requêtes réseau ("NetworkRequest"), etc. Finalement, nous avons la classe de vulnérabilités. Dans notre cas, la seule vulnérabilité à avoir été modélisée est celle des injections SQL ("SQLInjectionVulnerability"). Les axiomes de classes sont composés de 82 axiomes de sous-classe, de 23 classes équivalentes et de 14 classes disjointes. Les propriétés objet de l'ontologie se divisent en trois catégories. Il y a tout d'abord les propriétés permettant de modéliser l'environnement telles que "hasAdminRightsOn", "hasGroup" et "hasOperatingSystem". Ensuite, on retrouve les propriétés qui relient les événements aux éléments d'environnement telles que "hasDestination", "hasSource" et "happenedOn". Finalement, nous avons quelques propriétés caractérisant les événements telles que "hasCompletionStatus", "hasOperationType" et "hasEndingEvent". Les propriétés de données sont principalement utilisées par les pilotes pour ajouter aux éléments bruts leurs propriétés. Nous y retrouvons par exemple les

propriétés “hasIPAddress”, “hasFileName”, “hasPath”, “hasTimestamp”, “hasDestination-Port”, etc. Nous retrouvons finalement 28 règles SWRL associées à l’ontologie. Par exemple, la règle “T004” permet de relier un événement à sa partition en se basant sur le nom de la partition et sur la machine hôte. Un autre exemple est la règle “T013” qui permet de relier un événement à sa machine source en se basant sur l’adresse IP source de l’événement et les adresses IP des machines.

4.1.2 Artefacts de la modélisation du contrôle aérien

Le premier artefact obtenu par le processus de modélisation du contrôle aérien est une liste des senseurs disponibles pour le peuplement de l’ontologie. Ces sources sont en fait les différents types de radars existants dont l’information est divisée en sujets. Nous avons par exemple le sujet “Position PSR” qui représente de l’information obtenue des radars primaires. Les données de ce sujet sont relatives aux véhicules aériens en vol et sont composées de l’azimut, de la distance avec le radar, ainsi que du moment où la mesure a été enregistrée. Comme autres sujets, nous retrouvons aussi l’information récupérée par les radars secondaires et par les antennes ADS-B. Plusieurs autres sujets concernent des données de gestion du contrôle aérien, comme les demandes de changement de route, les plans de vol, etc. Contrairement au processus de la sécurité informatique, où les senseurs étaient définis à mesure qu’on tentait de répondre aux requis du système, nous les avons ici défini en amont.

Ensuite, nous avons défini une série de requêtes auxquelles le système devait pouvoir répondre en se basant sur l’information obtenue des senseurs précédemment définis. Nous avons obtenu deux types de questions, soit des questions d’ordre général et des questions reliés à des problèmes de sécurité. Nous retrouvons dans la première catégorie des questions telles que “Est-ce qu’il y a un risque de collision entre deux véhicules ?”, “Est-ce qu’un véhicule passe dans une zone restreinte ?”, “Est-ce qu’une route a été interrompue ?”, etc. La deuxième catégorie contient des questions comme “Est-ce qu’un véhicule viole une ou des lois de la physique ?”, “Est-ce qu’il y a la présence d’un véhicule fantôme ?”, “Est-ce que le système est attaqué par une attaque DoS ?”, etc. Nous avons aussi défini des hiérarchies de questions pour mieux définir les buts recherchés et les manières de calculer une réponse aux questions. Par exemple, la question qui cherche à savoir s’il y a un risque de collision entre deux véhicules consiste en fait en le calcul de l’intersection de deux zones de séparation, ce calcul étant lui-même en fait le calcul de l’intersection de deux cylindres. En effet, on détecte un risque de collision lorsque les “zones de séparation” de deux véhicules entrent en contact. Ces zones sont des cylindres virtuels définis autour des véhicules en fonction de leurs caractéristiques.

À partir de là, notre processus de recherche peut se diviser en deux. En effet, la première

partie du processus a été exécutée lors de la conception de la méthode et elle n'a donc pas généré les artefacts résultants de l'application de sa version finale, car elle se basait sur les méthodes de développement existantes. La première partie du processus a donc généré un champ lexical du contrôle aérien, quelques requêtes SPARQL découlant des questions précédemment spécifiées et, pour chaque requête, une modélisation sous la forme de graphes orientés. Ces graphes ont par la suite été traduits en ontologie. Cette ontologie contient des classes telles que "NamedPosition" pour les positions nommées par une étiquette et "ObservedPosition" pour les positions provenant des radars mais n'ayant pas d'étiquette, "Cylinder" pour le calcul des collisions, "Circle" pour le calcul des accès interdits, etc. L'ontologie créée dans cette première partie a été réfutée par des experts du domaine. Cela a impliqué la modification de la méthode de développement, ce qui a enclenché la deuxième partie du processus. Celle-ci a consisté à appliquer ATOM aux questions du contrôle aérien formulées précédemment, mais à cause de contraintes de temps, elle n'a pas été appliquée dans son entièreté. Les artefacts produits sont donc une ontologie, un diagramme de traduction et un document de spécification qui sont partiels, c'est-à-dire qu'une seule des questions précédemment formulées a été traitée. Certains problèmes de l'ontologie de la première étape du processus ont été réglés. Par exemple, le positionnement est maintenant modélisé avec les classes "Location", "Point" et "Position" pour lesquelles ont été introduites des variantes informationnelles relatives à la sécurité, soit "ObservedPosition", "ReportedPosition", "ObservedLocation", "ReportedLocation", etc. Cela a permis d'intégrer dans l'ontologie la notion de confiance envers l'information. Nous concluons de cela que ATOM est plus appropriée que les méthodes sur lesquelles nous nous basions pour faire le développement initial.

En résumé les artefacts produits en ce qui concerne le contrôle aérien sont une liste de questions en langage naturel, deux ontologies et un diagramme de traduction comportant une seule question. La première ontologie produite comporte 44 classes, 32 propriétés objet, 34 propriétés de données et cinq individus, encodées sous la forme de 382 axiomes. La deuxième ontologie comporte 31 classes, 16 propriétés objet et 10 propriétés de données, encodées sous la forme de 124 axiomes. Nous ne pouvons considérer aucune de ces ontologies comme étant prêtes à l'utilisation dans un système expert.

4.2 Quantification de l'effort total

On présente dans le Tableau 4.1 les données nous permettant d'évaluer l'efficacité et la pertinence du processus. Pour ce faire, nous quantifions le travail en nombre de séances de travail et en heures-personnes. Les nombres de séances nous permettent d'évaluer la charge organisationnelle et la répartition du travail dans le temps. Cependant, étant donné que les séances

ont été de diverses durées et qu'elles ont impliqué un nombre variable de personnes, il est difficile de les utiliser pour approximer les coûts réels du développement. Ainsi, nous présentons aussi le travail en heures-personnes, ce qui nous permet d'avoir une vision d'ensemble plus détaillée.

Tableau 4.1 Résultats du processus

		Technologie de l'information	**Contrôle aérien**
Nombre de séances	Plus d'une personne	77	12
	Une personne	7	28
	Total	84	40
Heures-personnes	Experts	435	102
	Ontologiste	373	202
	Total	708	304

Concernant les séances de travail, leur nombre de participants est allé d'une personne jusqu'à un maximum de six. Pour l'ontologie des technologies de l'information, nous avons eu un total de 84 séances de travail, dont 77 avec plus d'une personne impliquées et 7 avec une seule personne, soit plus que 10 fois plus de séances en équipe qu'en solo. En considérant un maximum d'une rencontre par journée ouvrable, cela représente une durée totale d'environ 17 semaines, soit 4 mois et une semaine. Pour le contrôle aérien, nous avons eu un total de 40 séances de travail, dont 12 en équipe et 28 en solo. Suivant les mêmes a priori que précédemment, cela représente 8 semaines, soit 2 mois de travail. Dans ce cas-ci, il y a eu plus que deux fois plus de séances en travail solitaire que de séances en équipe. Il faut noter ici que la plupart des séances à une personne étaient en fait des séances de travail de l'ontologiste. On peut aussi remarquer que nous avons eu deux fois plus de rencontres pour les technologies de l'information que pour le contrôle aérien. Cela peut s'expliquer par deux raisons. La première est une plus grande disponibilité de l'expertise des technologies de l'information par rapport à celle du contrôle aérien. La deuxième est une plus grande expertise et un plus grand intérêt de l'ontologiste pour les technologies de l'information que pour le contrôle aérien.

Concernant les heures-personnes, nous les avons divisées pour chaque ontologie en deux catégories : les heures-personnes des experts du domaine et les heures-personnes de l'ontologiste. Pour l'ontologie des technologies de l'information, 435 heures-personnes d'expertise en sécurité informatique ont été utilisées et 373 heures-personnes d'expertise en ontologie, pour un total de 708 heures-personnes. Pour l'ontologie du contrôle aérien, nous avons utilisé 102 heures-personnes d'expertise en contrôle aérien et 202 heures-personnes d'expertise en onto-

logie, pour un total de 304 heures-personnes. Nous tirons deux constats de ces données. Tout d'abord, la méthode développée est plus adaptée aux problèmes en technologie de l'information plutôt qu'en contrôle aérien étant donné qu'on a passé au total plus de deux fois plus de temps sur le développement de l'ontologie des technologies de l'information que sur celui de l'ontologie du contrôle aérien. Ensuite, nous avons utilisé beaucoup plus d'heures d'expertise du domaine pour l'ontologie des technologies de l'information que pour le développement de l'ontologie du contrôle aérien. Dans le cas des technologies de l'information, le nombre d'heures-personnes d'expertise représente 61% du nombre total alors que pour le contrôle aérien, il n'en représente que 34%. Étant donné que la première ontologie développée pour le contrôle aérien n'était pas appropriée, ces résultats semblent montrer qu'il est préférable d'enseigner la modélisation ontologique à des experts du domaine plutôt que l'inverse.

4.3 Métadonnées pour la sécurité informatique

Comme nous l'avons dit précédemment, le processus n'a été suivi complètement que pour la modélisation des environnements informatiques. Pour cette raison, cette partie du processus est particulièrement pertinente pour l'évaluation de ATOM. Nous présentons donc dans cette section des données obtenues du processus qui nous permettent d'en faire l'évaluation et de répondre à certaines de nos questions de recherche.

4.3.1 Division par scénarios d'attaques informatiques

Nous commençons par présenter des données qui nous permettent de répondre à notre quatrième question de recherche s'appuyant sur l'hypothèse qui veut qu'au fur et à mesure que le processus avance, le temps de développement devrait diminuer en raison de la réutilisabilité que permet l'ontologie. Pour ce faire, nous devons tout d'abord prouver que le temps de développement diminue au fil du processus. Nous commençons donc par présenter l'effort en fonction de chaque scénario en heures-personnes. Pour valider cette hypothèse, nous devons cependant prouver que la complexité des trois scénarios est semblable. C'est pour cette raison que nous présentons par la suite des données permettant de comparer leur complexité. Finalement, si une tendance négative est observée, nous devons démontrer qu'elle l'est en raison de la réutilisation des concepts précédemment introduits et non pas en raison d'autres facteurs. Nous finissons donc cette section en présentant des données concernant la réutilisation effective des deuxième et troisième scénarios.

L'effort en fonction des scénarios

Nous présentons maintenant les temps de développement divisés par scénarios d'attaques dans le contexte des technologies de l'information. Le Tableau 4.2 et la Figure 4.1 présentent les temps de développement en fonction des scénarios.

Tableau 4.2 Nombre d'heures en fonction du scénario

Scénario	Temps (heures)
1	87
2	59
3	54.5

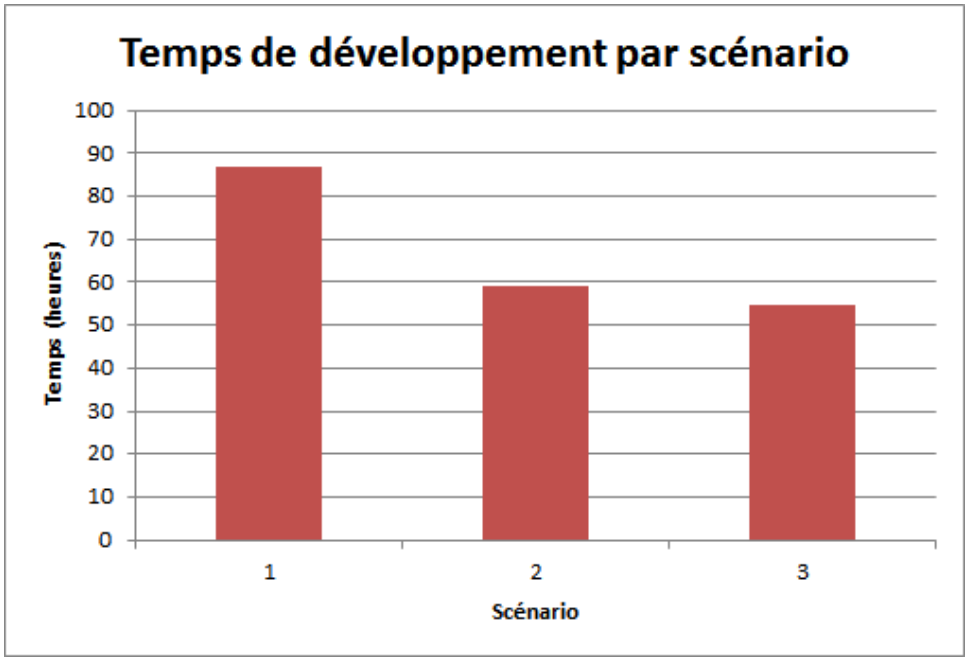


Figure 4.1 Graphique du nombre d'heures en fonction du scénario

Ici, nous voyons une grande diminution du temps entre le premier scénario et le deuxième et une petite diminution entre le deuxième et le troisième. À première vue, cela tend à répondre positivement à notre quatrième question de recherche au sujet d'un retour d'investissement des modélisations dans un domaine spécifique.

On pourrait se demander comment nous avons pu obtenir un total de 200.5 heures pour notre comptabilisation des heures par scénarios alors que nous obtenons un total de 708 heures pour la modélisation des technologies de l'information à la section 4.2. Cela peut s'expliquer principalement pour deux raisons. La première est que pour plusieurs séances de

travail, il est impossible de conclure que nous travaillions sur un scénario en particulier. Nous n'avons donc pas enregistré ce temps dans la division de l'effort par scénarios. La deuxième est que nous avons exclu l'étape d'enrichissement, puisqu'elle concerne l'ontologie dans son ensemble et non pas des scénarios en particulier. Il n'aurait pas été pertinent d'ajouter le temps d'enrichissement total à chaque scénario puisque ce qui nous intéresse au final est la comparaison entre ceux-ci.

Complexité des scénarios

Pour évaluer la complexité relative des 3 scénarios modélisés, nous nous basons sur des métriques relatives au diagramme de traduction. Nous faisons l'hypothèse que plus le diagramme d'un scénario comporte d'éléments, plus ce scénario est complexe. Nous supposons aussi que deux scénarios ayant un nombre d'éléments semblables sont de complexité semblable. Nous présentons ces résultats dans le Tableau 4.3.

Tableau 4.3 Évaluation de la complexité des scénarios

	Scénario 1	**Scénario 2**	**Scénario 3**
Nombre d'étapes	5	5	5
Nombre de pilotes	8	7	7
Nombre de règles de traduction	29	23	21
Nombre de requêtes intermédiaires	27	21	18

On conclut de ces données que les scénarios sont de complexités proportionnelles, bien que la complexité des scénarios semble légèrement diminuer à mesure qu'on avance dans ceux-ci, c'est-à-dire que le premier semble plus complexe que les deuxième et troisième scénarios et le deuxième semble plus complexe que le troisième. Nous pouvons cependant citer deux limitations sur cette métrique. En effet, les éléments qui indiquent cette diminution de complexité sont le nombre de requêtes intermédiaires et le nombre de règles de traduction. Dans le cas des requêtes, leur choix est plutôt arbitraire et une équipe de travail différente aurait pu donner des nombres complètement différents. Ensuite, un nombre décroissant de requêtes intermédiaires et de règles de traduction peut aussi s'expliquer par une meilleure compréhension du processus à mesure qu'on suit celui-ci, ce qui implique une plus grande efficacité dans l'enchaînement des requêtes et des règles. Nous pouvons cependant tout de même constater que la complexité des scénarios est dans un même ordre de grandeur.

Réutilisation des éléments de scénarios

Afin de valider notre hypothèse qui veut que le temps de développement devrait diminuer au fur et à mesure qu'on avance dans la modélisation des scénarios, nous présentons aussi les données de réutilisation des éléments du diagramme de traduction pour les deuxième et troisième scénarios dans le Tableau 4.4, le premier scénario servant de base à la réutilisation.

Tableau 4.4 Réutilisation des concepts

	Scénario 2		**Scénario 3**	
	Réutilisés	Total	Réutilisés	Total
Pilotes	4	7	4	7
Règles de traduction	12	23	7	21
Requêtes intermédiaires	5	21	1	18

On remarque de ces résultats que le scénario 2 réutilise plus d'éléments que le scénario 3. Pour confirmer notre hypothèse de la réutilisation impliquant une diminution du temps de développement, il aurait fallu que le scénario 3 présente une plus grande réutilisation. En effet, ce dernier avait les éléments des deux premiers scénarios dans lesquels puiser, alors que le deuxième n'avait que le premier scénario. Cela semble indiquer que la nature du deuxième scénario est plus semblable au premier et que le troisième est de nature différente. Ce résultat invalide l'hypothèse de diminution du temps de développement d'un scénario à l'autre en raison de la réutilisabilité possible. On conclut que la diminution du temps de développement est fortement influencée par d'autres facteurs, comme par la nature des scénarios.

4.3.2 Simulation du comportement du système expert

Une manière de valider que l'utilisation de ATOM permet l'implémentation d'un système expert fonctionnel a été de simuler manuellement le comportement du système expert en créant des instances, en exécutant un raisonneur sur l'ontologie, puis en l'interrogeant avec les questions servant de spécification au système. Pour illustrer cela, nous présentons ici quelques exemples. Nous n'explicitons pas les règles de traduction car la validation ne le demande pas. Il suffit de démontrer qu'après leur exécution, les requêtes retournent les résultats attendus en fonction des entrées brutes des senseurs.

Détection de trafic malveillant

Pour détecter le premier scénario, le système expert doit à un moment déterminer s'il y a eu du trafic potentiellement malveillant impliqué entre deux machines. Le diagramme de traduction de cette étape est illustré à la Figure 4.2.

Le comportement des pilotes des senseurs impliqués ainsi que de la règle "T014" est simulé manuellement par la création des instances "sc1_linux_machine", "sc1_windows_machine", "sc1_IRCRequest" et "sc1_hackerMachine". En créant toutes ces instances, nous simulons un trafic malveillant hypothétique et cela devrait donc, après l'application des règles de traduction "T008", "T009", "T010", "T011", "T012", "T013", "T015" et "T039", être détecté par la requête "R010"(4.1).

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX : <http://www.polymtl.ca/ontologies/siem#>
SELECT * WHERE {
     ?event rdf:type :MalwareTraffic .
     ?event :hasSource ?machine1 .
     ?machine1 rdf:type :Machine .
     ?event :hasDestination ?machine2 .
     ?machine2 rdf:type :Machine .
}
```
(4.1)

Cette requête consiste à récupérer tous les événements de type "MalwareTraffic" avec les machines impliquées, soit la machine source et la machine de destination du trafic. Dans notre cas, nous obtenons deux résultats impliquant la requête "sc1_IRCRequest" que nous avons spécifiée comme ayant "10.1.2.3" comme adresse IP source et "96.20.1.2" comme adresse IP de destination. La première adresse appartient aux machines "sc1_linux_machine" et "sc1_windows_machine" et la deuxième appartient à l'instance "sc1_hackerMachine". La requête "R010" retourne donc un résultat de trafic malveillant entre les machines "sc1_hacker Machine" et "sc1_linux_machine" et un résultat entre les machines "sc1_hackerMachine" et "sc1_windows_machine". L'obtention de ces résultats valide donc la procédure de traduction pour cette requête.

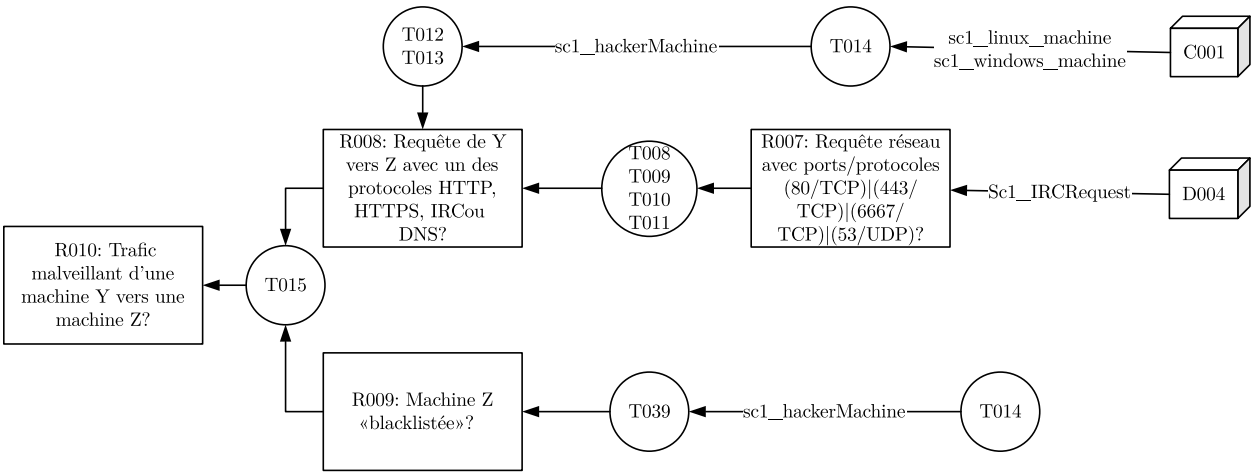


Figure 4.2 Diagramme de traduction de la détection de trafic malveillant

Accès à une page d'administration d'un site web

Une étape de détection du deuxième scénario consiste à savoir si une machine a effectué une requête à une page d'administration d'un site web. Le diagramme de traduction permettant au système expert de répondre à cette interrogation est présenté à la Figure 4.3.

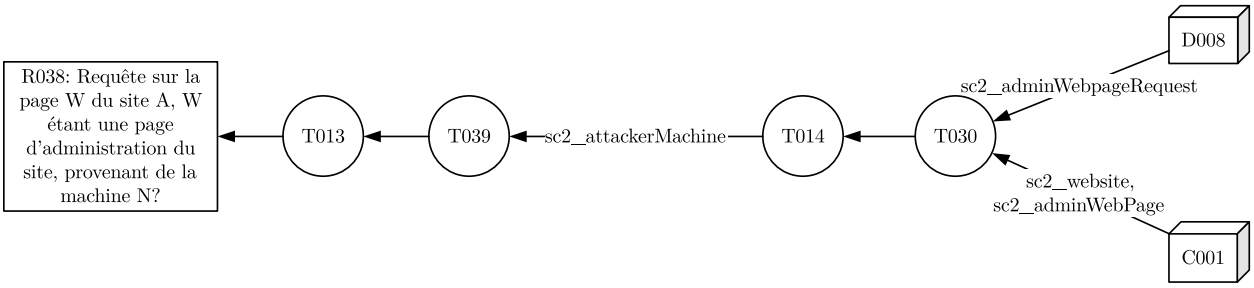


Figure 4.3 Diagramme de traduction de la détection d'un accès à une page d'administration d'un site Web

Elle implique deux senseurs, un qui crée une requête HTTP, “sc2_adminWebpageRequest”, et un qui crée les éléments d'environnement “sc2_website” et “sc2_adminWebPage”. Basé sur cette requête HTTP, la règle “T014” crée l'instance de la machine qui effectue la requête, “sc2_attackerMachine”. Ces instances sont créées manuellement dans l'ontologie de manière à ce que les règles de traduction “T013”, “T030” et “T039” puissent s'appliquer et que le système puisse répondre à la requête “R038”.

La requête “R038” (4.2) consiste à récupérer toutes les requêtes HTTP qui ont pour desti-

nation une page qui a été classifiée comme une page d'administration.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX : <http://www.polymtl.ca/ontologies/siem#>
SELECT * WHERE {
     ?event rdf:type :HTTPRequest .
     ?event :hasSource ?machine .
     ?event :hasDestination ?webpage .
     ?website :hasWebpage ?webpage .
     ?webpage rdf:type :AdminWebpage
}
```
(4.2)

Lorsque nous exécutons cette requête, nous obtenons un résultat qui indique que la machine "sc2_attackerMachine" a fait une requête à la page "sc2_adminWebpage" du site "sc2_website", ce qui démontre la validité de la procédure.

Détection d'un balayage de port sur un serveur

La première étape du troisième scénario consiste à détecter si un balayage de port a été effectué sur un serveur de l'environnement informatique analysé par le système expert. Le diagramme de traduction permettant la réponse à cette question est présenté à la Figure 4.4.

Le diagramme indique qu'un pilote de senseur crée un serveur "sc3_server" et son service "sc3_vpnService" et qu'un autre crée un événement "sc3_portScan". Ces éléments sont créés manuellement dans l'ontologie pour simuler le travail de ces pilotes. Suite à l'exécution des règles "T012", "T041" et "T042", la requête "R046" doit retourner un résultat. Cette requête est écrite en SPARQL dans l'énoncé 4.3.

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX : <http://www.polymtl.ca/ontologies/siem#>
SELECT * WHERE {
     ?event rdf:type :PortScan .
     ?event :hasDestination ?machine .
     ?machine rdf:type :Server
}
```
(4.3)

Dans notre cas, l'exécution de cette requête retourne ce qui était attendu, soit l'événement "sc3_portScan" et la machine "sc3_server".

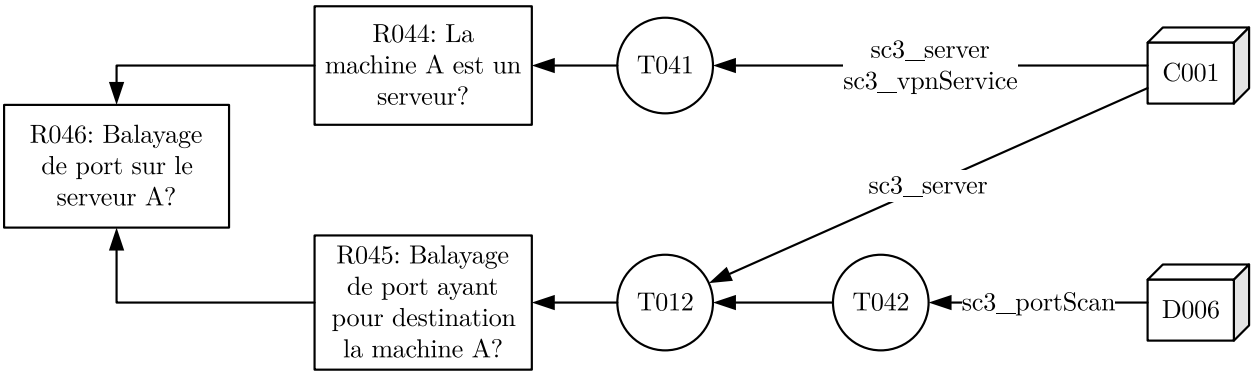


Figure 4.4 Diagramme de traduction pour la détection d'un balayage de ports

4.3.3 Qualité de l'ontologie de la sécurité des environnements informatiques

Pour évaluer la qualité de l'ontologie, nous avons utilisé *OOPS!*, un "scanner" de pièges de modélisation ontologique (Poveda-Villalón et al., 2012). Cet outil permet d'évaluer l'ontologie sur cinq dimensions : la compréhensibilité par l'être humain, la consistance logique, l'utilisation pertinente et efficace des vocabulaires et l'utilisabilité par des agents logiciels. Une sixième dimension non couverte par l'outil est la validité de la représentation du domaine. Étant donné que l'ontologie évaluée a été créée avec le concours d'experts du domaine, nous pouvons considérer que notre ontologie peut être évaluée positivement sur cette dimension.

L'outil *OOPS!* retourne les problèmes en les classifiant comme étant "critiques", "importants" ou "mineurs". Dans l'évaluation de notre ontologie, nous n'avons obtenu aucun cas critique. Comme cas importants, nous avons quatre absences de domaine ou de portée et le fait qu'aucune licence n'est déclarée. Comme cas mineurs, nous obtenons 117 cas d'annotations manquantes, 15 relations inverses non spécifiées et l'utilisation de conventions de nommage différentes. Nous soulignons que mis à part l'inexistence d'une licence et le manque d'uniformité dans nos conventions de nommage, les autres problèmes auraient pu être réglés dans la phase d'enrichissement et ne sont pas causées par ATOM, mais plutôt par notre manque d'expérience et notre manque de rigueur dans cette phase.

CHAPITRE 5 DISCUSSION

5.1 Réponse à nos questions de recherche

Dans cette section, nous présentons les réponses que nous avons apportées à nos questions de recherche ainsi que leur portée.

5.1.1 Question 1 : Étapes d'une méthode de développement ontologique

L'objectif principal de notre travail était de proposer une méthode de développement ontologique utilisable dans un contexte de système expert de détection d'anomalies et d'intrusions en sécurité informatique. Notre première question est directement reliée à cet objectif puisqu'elle demande quelles devraient être les étapes de cette méthode.

Le travail de modélisation appliqué à deux domaines de la sécurité accompli durant notre processus de recherche nous a permis de cibler certains problèmes des méthodes existantes, que nous avons présentés à la section 2.2.4, et de proposer une séquence d'étapes appropriées pour une méthode qui réponde à ces problématiques dans notre contexte. Nous la présentons au chapitre 3.

Nous avons démontré dans la section 4.3.2 que cette méthode génère des artefacts qui permettent l'implémentation d'un système expert fonctionnel en simulant le travail des senseurs et en questionnant le système après qu'il a exécuté ses règles de raisonnement. À ce sujet, nous pouvons conclure que le système répond à toutes les questions avec succès. C'est sur cette base que nous considérons que la spécification du système est assez complète pour son implémentation. Cependant, nous soulevons que cette méthode d'évaluation est fortement biaisée. En effet, puisque nous avons simulé le travail des pilotes des senseurs du système, nous avons en quelque sorte contrôlé les résultats de la méthode. Cela a pour conséquence que cette évaluation ne pouvait être que positive. Cependant, elle démontre que la méthode permet la création d'un système expert se comportant de manière correcte en théorie. Pour une évaluation plus rigoureuse, il aurait fallu implémenter le système dans son entier, y compris tous les pilotes nécessaires, et l'utiliser dans un environnement réel ou sur des données de test. Mais cela aurait été trop coûteux dans le cadre d'un mémoire de maîtrise.

Bien que nous n'ayons pas testé l'implémentation directement, nous nous appuyons sur notre expérience personnelle pour affirmer que la prochaine étape du processus est le développement à proprement dit du système. En effet, contrairement à d'autres méthodes de développement d'ontologies, nous explicitons clairement l'utilité de chaque concept introduit et de quelle

manière il est utilisé dans la résolution de problème. Le développeur de l'ontologie peut ainsi clairement présenter les possibilités de son ontologie et de quelle manière elle doit être utilisée. Un développeur d'application pourra prendre les documents produits par la démarche et implémenter un système fonctionnel à partir de ceux-ci puisqu'ils précisent tout le flot d'information, de sa génération par des senseurs à son utilisation concrète par l'utilisateur du système. Pour cette raison, nous considérons avoir répondu positivement à cette question de recherche.

5.1.2 Question 2 : Exigences d'une méthode de développement ontologique

Notre deuxième question de recherche consistait à demander quelles exigences on devrait appliquer à une méthode de développement d'ontologie de manière à ce qu'elle soit pertinente, c'est-à-dire qu'elle soit efficace, reproductible, facile à apprendre et à mettre en oeuvre. Au fil du processus, nous avons isolé cinq exigences que nous considérons nécessaires pour toute méthode de développement ontologique. Nous les avons présentées dans la section 3.1. Dans ce qui suit, nous rappelons ces exigences et les appliquons à ATOM.

Tout d'abord, nous voulions que la méthode crée des artefacts permettant l'implémentation d'un système expert. Cette exigence est en fait directement reliée à notre première question de recherche présentée à la section 5.1.1, que nous considérons répondue positivement.

Ensuite, une exigence visait à répondre aux problèmes de modélisation d'une ontologie qui peuvent être une cause de discussions prolongées produisant peu ou pas de valeur. Nous croyons que trois idées présentées par ATOM permettent d'éviter ces problèmes. La première consiste au rôle concret du système expert supporté par l'ontologie, soit la traduction du niveau d'abstraction de l'information d'un niveau plus proche de la machine à un niveau plus proche de l'utilisateur. La deuxième idée est de diriger le processus de modélisation par une série de requêtes SPARQL qui font office de spécifications au système. Ainsi, ce qui importe n'est pas d'obtenir une modélisation la plus "exacte" possible ou la plus maintenable, mais bien de permettre au système de répondre aux requêtes spécifiées. Finalement, la troisième idée consiste à faire un choix qui se rapproche le plus possible du langage de l'expert du domaine. En combinant ces trois idées, nous affirmons que l'ontologiste a la plupart du temps tous les outils nécessaires pour trancher sur les possibilités de modélisation, et cela de manière justifiée. Nous considérons donc que l'exigence de donner des moyens d'accélérer le développement est atteinte.

La troisième exigence consistait à pourvoir un processus avec des étapes claires permettant de diriger le processus de création de l'ontologie en entier. De manière superficielle, cet objectif a été atteint puisqu'on présente une méthode en six étapes qui produit des artefacts de spéci-

fications d'un système expert de détection d'intrusion, y compris une ontologie. Cependant, nous ne pouvons pas affirmer si la méthode est assez détaillée ou trop restrictive.

Par la suite, nous avions comme exigence la production d'une ontologie de bonne qualité. La cinquième étape de ATOM qui consiste à l'enrichissement de l'ontologie permet de répondre à ce critère de qualité. Nous avons présenté une évaluation de la qualité de l'ontologie des environnements informatiques dans la section 4.3.3. Bien que dans notre cas, quelques lacunes de qualité aient pu être constatées, cela n'est pas la conséquence d'un problème de ATOM mais plutôt de notre manque d'expérience de modélisation ou de notre manque de rigueur au moment de la dernière étape. De plus, il faut noter que ATOM produit des artefacts améliorant la réutilisabilité des ontologies développées. En effet, chaque concept a une place claire et définie dans un processus décisionnel plus large, ce qui permet de saisir une définition plus complète des concepts de l'ontologie, comme définis par les concepteurs de l'ontologie, et ainsi de les réutiliser de manière pertinente. Étant donné que la réutilisabilité d'une ontologie contribue à sa qualité, nous croyons que cela contribue aussi à l'atteinte de notre objectif de qualité.

Notre dernière exigence consistait en ce que la méthode implique des temps de développement raisonnables pour son utilisation en industrie. Comme nous l'avons mentionné, cette évaluation est très relative à la valeur marchande du système expert produit. Cependant, à la lumière de la quantification de l'effort total en heures-personnes de notre section 4.2, nous jugeons que ATOM semble anormalement coûteuse. En effet, 708 heures-personnes ont été utilisées pour le développement de la spécification du système expert, et cela représente le travail effectué pour seulement trois scénarios. Cela nous semble élevé pour un travail de spécification. De plus, l'hypothèse de réutilisabilité ne semble pas exacte puisque que nos données semblent démontrer que les temps de développement sont fortement influencés par la nature des exigences du système, nature difficilement caractérisable. Nous ne pouvons donc pas compter sur cette diminution de temps de développement. Nous pouvons aussi énoncer les 304 heures-personnes du développement de l'ontologie du contrôle du trafic aérien qui n'ont abouties au final à aucun résultat utilisable. Cette conclusion est cependant discutable puisque dans la quantification de l'effort, nous avons aussi compté le développement de la méthode qui a évolué au fil du processus. Nous n'avons pas non plus fait de comparaison avec des méthodes semblables pour faire un différentiel de temps et nous ne pouvons donc pas affirmer hors de tout doute que nos temps de développement sont anormaux pour ce genre de travail.

En conclusion, nous répondons clairement à trois de nos exigences, soit celles de la documentation d'un système expert, de la perte de temps de modélisation et de la qualité de

l'ontologie. Nous ne pouvons pas clairement affirmer notre atteinte de l'exigence de l'explicitation du processus, puisque bien qu'il soit suffisant à nos besoins, il pourrait ne pas l'être à d'autres. Finalement, nos résultats semblent démontrer que ATOM n'est pas viable en ce qui concerne le temps de développement. Cependant, cette conclusion est fortement biaisée sur nos propres expériences et nécessiterait une validation plus formelle.

5.1.3 Question 3 : Flexibilité de la méthode en sécurité

Notre troisième question demandait si une méthode de création d'ontologie spécifique à l'implémentation d'un système expert en sécurité serait applicable à plusieurs domaines de la sécurité. Afin de pouvoir répondre à cette question, nous avons travaillé à la modélisation de deux domaines de la sécurité : la détection d'intrusions dans les environnements informatiques et la détection d'anomalies en contrôle du trafic aérien.

L'application complète de la méthode à ces deux domaines nous aurait permis de répondre à cette question. Malheureusement, en raison de la nature de notre processus de recherche, nous n'avons pas pu appliquer complètement notre méthode au contrôle du trafic aérien. En effet, nous avons créé deux ontologies partielles qui ne sont pas suffisamment matures pour être utilisées dans un système expert. Pour cette raison, il ne nous est pas possible de répondre à cette question. Cependant, étant donné que la deuxième ontologie partielle créée avec une méthode plus mature a pu répondre aux problèmes de la première ontologie créée, nous considérons qu'une application complète de ATOM à ce domaine nous permettrait probablement de répondre positivement à cette question de recherche.

5.1.4 Question 4 : Retour d'investissement intra-domaine

Notre quatrième question de recherche demandait si nous allions avoir un retour d'investissement dans la modélisation d'un domaine particulier se manifestant sous la forme de diminutions des temps de modélisation au fur et à mesure de l'application du processus. Cette question se basait sur l'hypothèse de la réutilisation possible des modélisations effectuées précédemment dans le processus de développement.

De manière à pouvoir répondre à cette question, nous avons modélisé trois scénarios de la sécurité des environnements informatiques et avons enregistré les temps de développement pour chacun. Ces résultats sont présentés à la section 4.3.1. Bien que nous démontrions que les scénarios soient de complexité semblable et qu'on constate une diminution du temps de développement d'un scénario à l'autre, nous ne pouvons pas répondre positivement à la présente question de recherche. En effet, nous constatons une incohérence entre les données

de réutilisation des deuxième et troisième scénarios et l'hypothèse de la diminution du temps basé sur la réutilisation. En effet, si nous avions inversé le premier et le troisième scénario, nous n'aurions probablement pas obtenu une courbe de temps décroissante.

Pour expliquer ces résultats, nous pouvons émettre l'hypothèse qu'il existe probablement d'autres facteurs que la réutilisation qui influencent les temps de modélisation. Entre autres, nous pouvons énoncer la nature des scénarios que nous abordons plus en détail dans la section 5.4. On doit aussi considérer un biais d'apprentissage qui implique que nous ayons été de plus en plus à l'aise avec la modélisation au fur et à mesure de l'avancement du processus. Ensuite, il est probable que la méthode proposée dans ce travail ne soit pas bien adaptée à la réutilisation. Plus le nombre de modélisations augmente, plus il est difficile de savoir lesquelles sont réutilisables. Cet aspect dépend presque uniquement de la mémoire des développeurs de l'ontologie.

Nos résultats ne nous permettent donc pas de répondre clairement à cette question.

5.1.5 Question 5 : Retour d'investissement inter-domaine

Cette question est semblable à notre troisième question mais elle concerne les modélisations de domaines différents de la sécurité. L'idée ici était de chercher des "motifs" réutilisables dans les modélisations des domaines de la sécurité.

C'est aussi pour répondre à cette question que nous avons travaillé sur le contrôle du trafic aérien en plus des environnements informatiques. Cependant, comme nous l'avons mentionné lors de la réponse à notre troisième question de recherche (section 5.1.3), nous n'avons pas appliqué notre méthode en entier à la modélisation du contrôle du trafic aérien. Pour cette raison, nous ne pouvons rien conclure au sujet de la présente question.

5.2 Réflexion sur la place de l'ontologie dans le domaine de l'intelligence artificielle

Dans notre démarche, nous avons été appelés à nous questionner au sujet de la place de l'ontologie dans le domaine de l'intelligence artificielle. Particulièrement, nous avons cherché à répondre à trois questions :

1. De quelle manière l'ontologie doit-elle être utilisée dans un système expert pour qu'on le qualifie d'intelligent ?
2. De quelle manière peut-on utiliser l'apprentissage machine dans un système expert basé sur l'ontologie ?

3. Qu'est-ce qui distingue un système sémantique d'un système syntaxique et quelle est l'implication de l'ontologie dans cette distinction ?

Dans cette section, nous présentons des pistes de réflexion basées sur notre expérience de recherche pour répondre à ces questions.

5.2.1 Modes d'utilisation de l'ontologie

En analysant notre démarche, on peut s'interroger sur la présence d'intelligence artificielle dans notre système. En fait, nous remarquons que bien que la représentation de l'information par ontologie soit du domaine de l'intelligence artificielle, il est facile de se servir de l'ontologie simplement comme d'un graphe orienté sur lequel on peut faire des requêtes. Si l'on associe la partie "intelligente" du système aux règles logiques qui y sont encodées, alors nous n'utilisons cette intelligence que pour faire les traductions de l'information de manière à ce qu'elle soit plus accessible pour l'utilisateur. Cette utilisation est le résultat d'une analyse du travail de l'expert en sécurité. En effet, il cherche toujours à répondre à des questions assez spécifiques et, pour ce faire, il fait l'amalgame de toutes les informations qu'il peut trouver. C'est ce processus de construction mentale que nous avons voulu simuler avec l'ontologie dans le système expert proposé. Donc, notre utilisation de l'ontologie est fondée sur l'hypothèse que ce qui distingue l'expert de la machine est sa capacité à prendre une information très brute et à l'abstraire de manière à pouvoir répondre à des questions plus proches de son langage qui façonne ses modèles internes. Cela explique pourquoi nous proposons plusieurs méthodes de traduction : la logique descriptive, des règles logiques, des algorithmes quelconques, des modèles appris par apprentissage machine, etc. La logique possible encodée dans l'ontologie n'est qu'une manière parmi d'autres de faire le travail de traduction opéré mentalement par l'expert. Notre système n'est donc pas "intelligent" à proprement dit, mais il encode les connaissances, ou autrement dit l'intelligence, d'experts du domaine.

Le point précédent est important, car cette position a ses avantages et ses désavantages. Un problème apparent est tout d'abord la nécessité de spécifier explicitement cette intelligence, c'est-à-dire, d'encoder les processus de traduction explicitement dans le système. Nous sommes en effet beaucoup plus dans un paradigme opérationnel que déclaratif. Cela a pour conséquence principale que tous les résultats sont prévisibles et qu'ils peuvent donc être testés. Cela peut ne pas sembler un problème à première vue, mais une des raisons qui explique l'engouement récent pour l'intelligence artificielle est justement la possibilité de ne pas avoir à tout spécifier, puisque qu'il est impossible de le faire pour plusieurs problèmes, et d'obtenir de bons résultats. C'est particulièrement le cas des systèmes à apprentissage non supervisé, mais les systèmes supervisés peuvent aussi entrer dans cette catégorie. Par exemple, au lieu

de coder des algorithmes qui disent à un hélicoptère comment réagir en toutes situation en fonction des données de navigation, on crée un environnement de pilotage d'hélicoptère virtuel et on laisse l'algorithme apprendre de ses erreurs jusqu'à ce que son taux d'erreur soit pratiquement nul. La solution proposée dans ce travail n'entre pas dans cette catégorie.

Bref, d'une certaine manière, notre solution s'inscrit difficilement dans le domaine de l'intelligence artificielle puisque nous encodons de manière algorithmique la connaissance de l'expert, ce qui n'est pas très différent de la programmation classique. Le fait d'utiliser l'ontologie comme moyen de stockage de l'information dans un système ne le rend pas "intelligent". C'est la manière d'utiliser l'ontologie qui fait la différence. En effet, il est possible d'utiliser la capacité de raisonnement qu'elle permet par sa spécification en logique descriptive. Il est cependant aussi possible de traiter l'ontologie purement comme un graphe orienté, structure utile mais limitée en ce qui concerne le raisonnement automatisé. Nous avons trouvé que l'utilisation de SPARQL nous a égarés sur cette question. Lorsqu'on utilise ce formalisme d'interrogation, nous ne sommes pas très loin d'une interrogation classique d'une base de données. Avec ce genre d'interrogation, nous n'exploitons ni le paradigme du monde ouvert de l'ontologie ni la capacité de raisonnement qu'elle permet dans l'incertitude. Une approche différente qui nous aurait rapproché d'une certaine "intelligence artificielle" aurait été de générer nos questions en logique descriptive plutôt qu'en SPARQL.

Pour illustrer ce propos, prenons l'exemple que nous avons donné à la section 4.3.2, qui consiste à détecter un balayage de ports sur un serveur, et transformons-le de manière à ce que le raisonneur de logique descriptive puisse faire tout le travail de traduction et puisse répondre à notre requête. Commençons par traduire la requête SPARQL finale en logique descriptive. La requête "R046" est présentée dans l'énoncé 5.1.

```
SELECT ?event WHERE {
    ?event rdf:type :PortScan .
    ?event :hasDestination ?machine .
    ?machine rdf:type :Server
}
```
(5.1)

On la traduit en logique descriptive comme présenté dans l'énoncé 5.2.

$$PortScan \sqcap \exists hasDestination.Server \tag{5.2}$$

Ensuite, on traduit la seule règle SWRL de notre exemple, la règle "T012", qui consiste à

relier tous les événements réseaux à leur machine destination en se basant sur les propriétés de données “hasDestinationIPAddress” et “hasIPAddress”. Les étapes sont :

1. Relier l’événement et la machine par une seule entité d’adresse IP. Pour ce faire, on crée une nouvelle classe “IPAddress” pour modéliser les adresses IP à laquelle on associe une propriété de données “hasValue” qui contiendra l’adresse IP sous la forme d’une chaîne de caractères.
2. Spécifier que la propriété “hasValue” est la propriété “Target for Key” de la classe “IPAdress”, ce qui indique que si deux instances de cette classe ont la même valeur pour la propriété “hasValue”, alors ce sont la même instance.
3. Spécifier une propriété inverse à “hasIPAddress”, que nous appellerons “isIPAddressOf”
4. Spécifier que la propriété “hasDestination” a pour super-propriété la chaîne de propriétés “hasDestinationIPAddress” suivie de “isIPAddressOf”.

De cette manière, le travail de la règle SWRL sera entièrement accompli par le raisonneur en se basant uniquement sur des axiomes de logique descriptive.

Pour comprendre en quoi cette solution est supérieure à la précédente, supposons que plus tard, le développeur de l’ontologie décide qu’un service peut être la destination d’une requête réseau telle qu’un balayage de ports. Ainsi, il ajoute une nouvelle super-propriété à la propriété “hasDestination” qui implique que toute requête ayant pour destination un service a aussi pour destination le serveur qui offre ce service. Cela se traduit par la chaîne “hasDestination” suivi de “isOfferedBy”. Prenons maintenant le cas où un service a été instancié, mais pas le serveur qui l’offre. Étant donné que la classe “Service” est définie comme étant offerte (“isOfferedBy”) par un serveur, le raisonneur sait que le service doit nécessairement avoir un serveur même si cela n’a pas été spécifié et la requête “R046” formulée dans l’énoncé 5.2 retournera une réponse positive à propos du fait qu’un serveur a reçu un balayage de ports. Dans cette même situation, la requête écrite en SPARQL ne retournera aucun résultat puisqu’elle nécessite la création d’une instance de serveur. On peut donc conclure que la deuxième modélisation est plus “intelligente” puisqu’elle exploite les capacités logiques du raisonneur, contrairement aux requêtes SPARQL qui ne font qu’interroger des données suivant un “pattern” défini.

5.2.2 Complémentarité avec l’apprentissage machine

Bien que les modèles obtenus par les algorithmes d’apprentissage machine obtiennent d’excellents résultats dans plusieurs domaines, il semble qu’à long terme ses applications soient

limitées. On peut par exemple constater les ratés actuels de cette technologie pour la détection d'intrusion : "Despite the promising nature of anomaly-based IDS, as well as its relatively long existence, there still exist several open issues regarding these systems." (Hollander, 2017). Quelques-uns de ces problèmes sont de hauts taux de faux positifs, la facilité de tromper les modèles appris, ainsi que les vitesses de traitement insuffisantes des implémentations des modèles par rapport aux débits d'information à traiter. On pourrait argumenter que la technologie est encore jeune et que ces problèmes seront réglés dans le futur. Cependant, selon quelques auteurs, certains domaines nécessiteront toujours une interconnexion entre les systèmes dits logiques et ceux d'apprentissage machine. Yoav Hollander donne comme exemple le domaine de la vérification des systèmes autonomes intelligents (Garcia-Teodoro et al., 2009). Selon lui, une question de recherche importante dans le futur sera de trouver comment utiliser ces deux technologies de manière complémentaire.

Le cadre que nous proposons pour l'utilisation de l'ontologie en détection d'intrusion répond en partie à cette interconnexion entre les deux technologies. En effet, notre modèle général est basé sur une multitude de montées en abstraction. Les mécanismes de traduction peuvent être tout algorithme imaginable, y compris ceux exploitant des techniques d'intelligence artificielle. Nous pouvons donc exploiter les forces et faiblesses de chaque approche pour créer un système global plus "intelligent". On exploite la capacité d'abstraction et de raisonnement contrôlé que nous permet l'ontologie et la logique et on exploite la grande force de classification des modèles d'apprentissage machine à partir de données complexes pour lesquelles des règles seraient insuffisantes.

Illustrons cette idée par un exemple où l'apprentissage machine donne un bon taux de réussite, soit la classification d'images par résonance magnétique pour la détection du cancer chez un patient. On voit ici un exemple de règle de traduction qui prend des données très brutes, soit l'information stockée d'une image, et qui classifie cet amas d'information en information plus abstraite, soit la présence ou pas de cancer dans l'image. On peut alors imaginer un système expert stockant sous la forme d'une ontologie les profils de clients d'une clinique. Lors de la prise d'images par résonance magnétique, celles-ci pourraient être attachées virtuellement au profil du client et le modèle appris pourrait alors classifier le client selon les résultats d'analyse des images obtenues. Dans le cas d'une identification positive, le système pourrait proposer divers traitements en fonction du profil du client en se basant sur des règles logiques qui exploitent les types des profils. Cela est purement hypothétique puisque nous ne sommes pas très familiers avec le domaine de la santé. Cependant, cet exemple illustre bien comment la manière d'utiliser et de développer l'ontologie que nous proposons peut être complémentaire avec l'apprentissage machine. De plus, d'un point de vue du diagnostic, les médecins ne peuvent approuver la conclusion d'un algorithme sans comprendre comment il est arrivé à

cette conclusion. On peut seulement conclure de ces exemples que la complémentarité des deux technologies est possible et même souhaitable et notre solution permet de les utiliser toutes deux dans un seul cadre.

5.2.3 La sémantique des systèmes logiques

En général, on définit la sémantique comme étant “the study of the relation between strings and their meanings” (Van Eijck et Unger, 2010). En logique, il n’y a pas de manière évidente d’expliciter le “sens” exact d’un axiome. Cela s’explique par les diverses interprétations possibles des symboles non logiques utilisés dans ceux-ci. Par exemple, le symbole “PaysDémocratique” peut dénoter des objets différents selon l’interprétation du monde. Une vision possible de sa définition est la classe de tout ce qui est un pays où ont lieu des élections basées sur le vote de sa population. Mais certaines personnes pourraient argumenter que les élections de certains pays comportent des irrégularités qui en font des pays n’appartenant pas à la classe “PaysDémocratique”. Ainsi, lorsqu’on parle de sémantique en logique, on ne parle pas de l’aspect de la sémantique abordé par le dictionnaire. On parle plutôt d’une spécification du sens qui est “a function of the interpretation of the predicate and function symbols.” (Levesque, 1988) La sémantique d’un univers est donc capturée par une interprétation associée aux symboles du système. Par exemple, une des interprétations possibles de “PaysDémocratique” contiendra le Canada alors qu’une autre pourrait ne pas le contenir. Cette définition de la sémantique n’est utile qu’en ajoutant des clauses reliant logiquement les symboles entre eux. De cette manière, on réduit le nombre d’interprétations possibles et on peut tirer des conclusions qui simulent le raisonnement humain. Par exemple, si on définit la classe des “PaysDémocratique” comme étant disjointe des “PaysPeuDémocratiques” et qu’on définit les “PaysPeuDémocratique” comme étant les pays qui ont eu un processus électoral présentant des irrégularités, alors l’interprétation dans laquelle un tel pays est un “PaysDémocratique” ne sera pas valide.

À la lumière de cette définition de la sémantique, nous pouvons conclure que notre travail s’est peu penché sur cette facette de l’ontologie. Nous utilisons le raisonnement logique toujours à des fins concrètes et planifiées et nous le mettons au même niveau que les règles SWRL et des algorithmes quelconques, puisque notre but est toujours de permettre des requêtes plus abstraites. Pour cette raison, nous utilisons peu les définitions de sous-classes anonymes et d’équivalence dans nos définitions de classes. De même, nous utilisons peu les caractéristiques des propriétés (fonctionnelle, symétrique, etc.). Ces éléments sont ajoutés surtout à la phase d’enrichissement de la méthode de développement. Donc, d’affirmer que notre solution est “sémantique” plutôt que “syntaxique” est peu approprié. Il serait cependant intéressant

d'explorer une utilisation plus poussée de ces concepts. Par exemple, comme présenté dans la section 5.2.1, la traduction de nos requêtes SPARQL en requêtes formulées purement en logique descriptive et une traduction des règles SWRL en axiomes de logique descriptive serait un pas dans cette direction. En effet, l'orientation initiale de notre modélisation qui nous a poussés à utiliser SPARQL comme outil de requête nous a enfermés dans un mode de recherche syntaxique plutôt que sémantique. Si nous avions plutôt orienté notre travail sur des requêtes en logique descriptive, alors nous aurions plus facilement pu qualifier notre approche comme étant sémantique.

5.3 Impacts positifs sur la maintenance

Un point que nous voulons discuter est la difficulté de maintenir une ontologie et en quoi notre solution peut potentiellement faciliter cet exercice en introduisant des tests automatisés. Ce constat se base sur deux éléments de notre solution. Tout d'abord, étant donné que nos modélisations sont la conséquence d'un travail de réponse à des séquences de question, les résultats attendus sont relativement clairs et pourraient être sauvegardés. Ensuite, on peut s'attendre à ce que ces résultats soient toujours pertinents en raison de notre documentation partielle des "pilotes" qui consiste en des instances exemplifiant ce qu'ils doivent créer dans l'ontologie. Suivant les résultats déterministes des simulations présentées dans la section 4.3.2, les modifications apportées à l'ontologie ne devraient pas modifier les résultats de ces simulations, à moins d'une restructuration majeure. Il serait donc relativement facile de créer un outil exécutant une série de requêtes SPARQL et validant le résultat obtenu en utilisant le résultat attendu pour chacune d'elle. À chaque itération de maintenance, on pourrait donc utiliser cet outil sur la nouvelle version de l'ontologie et ainsi valider que le comportement initial est conservé.

Dans un cadre plus large, un autre avantage que notre solution apporte à la phase de maintenance est l'utilisation de la logique descriptive pour décrire les objets de notre univers. Ainsi, un raisonneur peut détecter les incohérences logiques dans l'ontologie, lors de la maintenance. Cela permet d'éviter des erreurs rapidement et d'éviter des coûts de réparation potentiels à long terme.

5.4 Difficulté de modélisation en fonction du domaine

Une conclusion que notre travail nous permet de tirer concerne la différence de modélisation entre deux domaines présentant des caractéristiques différentes. En effet, nous avons travaillé sur la modélisation de l'information relative aux technologies de l'information et à celle

relative au contrôle du trafic aérien. Dans le premier cas, nous étions confrontés à un monde hautement chaotique et non structuré, alors que dans le deuxième cas, nous avions un monde très structuré, régi par des lois naturelles et qui présentait même des notions sémantiques dans l'encodage actuel de l'information. Nous sommes donc à même de pouvoir tirer des conclusions quant au différentiel de difficulté entre les deux domaines.

Notre hypothèse était tout d'abord qu'il serait beaucoup plus facile de modéliser le domaine du contrôle du trafic aérien. Cependant, nous avons expérimenté l'inverse et, lorsque nous analysons les ontologies produites, les résultats sont plus convaincants en ce qui concerne les technologies de l'information plutôt que celui du contrôle du trafic aérien. En effet, nous n'avons ni spécification complète d'un système expert, ni ontologie complète pour le contrôle du trafic aérien. Mis à part le fait qu'un plus grand effort ait été consacré à l'ontologie des technologies de l'information, un élément contribuant majeur a été la répartition différente de l'expertise dans les deux cas. Pour le domaine des technologies de l'information, l'ontologiste avait déjà une expertise dans ce domaine et l'expert principal s'est bien formé sur les ontologies. Dans le cas du contrôle du trafic aérien, l'ontologiste a dû apprendre de zéro le domaine et les experts du domaine étaient moins bien formés aux ontologies. Ainsi, nous avons constaté qu'une plus grande structure du domaine a un impact négligeable par rapport à celui des expertises des intervenants. Cela tend à prouver qu'il est préférable de former un expert de domaine à l'ontologie plutôt qu'un ontologiste à un domaine.

Un autre point qui a contribué à la réfutation de l'hypothèse énoncée est que le domaine du contrôle du trafic aérien, qui paraissait de prime abord plus structuré, ne l'était pas suffisamment pour faciliter de manière significative la modélisation par rapport au domaine des technologies de l'information. En effet, une constatation est qu'il est très dur d'évaluer et de prédire l'effort de modélisation d'un domaine. La raison à cela est notre difficulté à systématiser la synthèse et la représentation de l'information. Par exemple, est-ce que l'information représentée sous la forme d'un tableau est plus simple à comprendre et à utiliser que cette information représentée sous la forme d'un graphe ? La réponse à cette question dépendra de plusieurs éléments, comme de la quantité et de la nature de l'information, de la manière de fonctionner des intervenants, de l'utilisation prévue de cette information et du médium d'affichage. Est-ce que le fait que le contrôle du trafic aérien soit régi par des lois naturelles, contrairement aux environnements informatiques, implique une plus grande facilité de modélisation ? Cela dépend encore une fois de l'utilisation. Si l'on veut modéliser rigoureusement les structures pour la détection de collision en contrôle du trafic aérien, bien que cela soit bien défini et très structuré, la modélisation risque d'être beaucoup plus complexe que n'importe quelle modélisation des technologies de l'information, où il sera la plupart du temps possible d'abstraire l'information.

CHAPITRE 6 CONCLUSION

Pour conclure, rappelons tout d'abord notre démarche. Nous sommes partis des différents problèmes des systèmes actuels de détection d'intrusion et sur l'hypothèse de Sadighian (2015), ainsi que d'autres auteurs, que la représentation de l'information par l'ontologie peut régler plusieurs de ces problèmes. Notre but était donc tout d'abord de concrétiser cette idée pour la rendre propice à l'implémentation. Ce faisant, nous avons constaté qu'il n'existait pas de méthode de développement d'ontologies adaptée à notre besoin, celui-ci étant d'obtenir une ontologie prenant place dans un système expert de détection d'intrusion ou d'anomalie. Nous avons donc cherché à répondre aux cinq questions suivantes :

1. Quelles devraient être les étapes d'une méthode de développement d'ontologies dans ce contexte ?
2. À quelles exigences cette méthode devrait-elle répondre ? Autrement dit, sur quels critères pourrait-on l'évaluer ?
3. Est-ce qu'une méthode développée dans ce contexte s'appliquerait à plusieurs domaines de la sécurité ?
4. Est-ce que cette manière de faire implique un retour sur investissement dans la modélisation d'un domaine spécifique ?
5. Est-ce que cette manière de faire implique un retour sur investissement dans la modélisation de plusieurs domaines différents de la sécurité ?

Nous avons travaillé à la modélisation de deux domaines de la sécurité dans le but de répondre à ces questions : le contrôle aérien et la détection d'intrusion dans les environnements informatiques.

Nous proposons donc tout d'abord ATOM, la méthode développée pour répondre à notre première question, en réaction aux difficultés de prédire l'utilisation de l'information et d'utiliser les ontologies existantes dans un contexte de système expert tel qu'expliqué à la section 2.2.4. Cette méthode est basée sur l'hypothèse de l'information qui se présente par niveaux d'abstraction et sur l'hypothèse que le rôle du système expert est de permettre la navigation entre ces niveaux. ATOM est aussi complètement centrée sur les besoins prédéfinis du système, c'est-à-dire, sur les requêtes auxquelles il devra répondre. Elle tranche sur des questions problématiques du développement de l'ontologie en portant son accent sur les formulations en langage naturel des experts de domaine. Finalement, elle fournit une documentation permettant l'implémentation concrète d'un système expert basé sur l'ontologie développée. ATOM se distingue des autres méthodes principalement en raison de son pragmatisme. En effet,

les artefacts produits par son application consistent en la spécification d'un système expert. Ainsi, il est possible de prendre ces documents et d'implémenter un système expert ayant pour schéma les ontologies développées. Contrairement aux méthodes existantes, chaque élément des ontologies développées par la méthode que nous proposons peut être justifié, et ce sur trois plans : son utilisation dans une réponse à une requête spécifiant le système, un rapprochement du langage de l'expert du domaine et la qualité générale de l'ontologie à des fins de compréhensibilité et de réutilisabilité.

Au cours du processus, nous avons isolé cinq exigences d'évaluation pour répondre à notre deuxième question. Celles-ci sont :

1. Produire des artefacts permettant l'implémentation d'un système expert fonctionnel.
2. Pourvoir des guides et directions permettant de justifier les choix de modélisation et d'être plus efficace.
3. Donner une direction claire composée d'étapes séquentielles à effectuer.
4. Produire une ou des ontologies d'une qualité suffisante pour son partage et sa réutilisation.
5. Être applicable dans une durée de temps raisonnable.

Bien que ces exigences aient découlé de notre processus de recherche et aient guidé le développement de la méthode que nous proposons, nous considérons n'avoir atteint que les deux premières et la quatrième exigences. On ne peut pas tirer de conclusion au sujet de la troisième exigence étant donné que nous n'avons utilisé la méthode que sur peu de cas. Puis, lorsqu'on analyse les résultats de temps obtenus pour les diverses modélisations effectuées présentés à la section 4.2, nous pouvons poser un doute sur l'atteinte de la quatrième exigence.

En ce qui concerne notre quatrième question de recherche, les résultats obtenus à la section 4.3.1 indiquent une diminution de temps au fur et à mesure que nous avancions dans les modélisations des environnements informatiques. Cependant, nous avons constaté que cette diminution n'était pas causée par la réutilisation des composantes. Notre hypothèse est que la nature des scénarios étudiés ainsi qu'un biais d'apprentissage des développeurs ont provoqué cette différence dans les temps de développement.

Finalement, nous n'avons pas pu répondre à nos troisième et cinquième questions de recherche puisque pour ce faire, il aurait fallu aller plus loin dans le processus de développement d'ontologies dans des domaines différents de la sécurité. En effet, bien que nous ayons travaillé sur deux de ces domaines pour répondre à ces questions, nous n'avons pas été suffisamment loin dans la modélisation du contrôle aérien pour pouvoir répondre de manière satisfaisante à ces questions.

Ensuite, nous avons eu une réflexion en ce qui concerne la place de l'ontologie dans le domaine de l'intelligence artificielle que nous présentons à la section 5.2. Dans cette réflexion, nous avons cherché à répondre à trois questions :

1. De quelle manière doit-on utiliser l'ontologie dans un système pour qu'il soit qualifié d'intelligent ?
2. Est-ce que l'ontologie utilisée conjointement avec l'apprentissage machine est possible et pourrait répondre à des besoins actuels ?
3. Que signifie l'affirmation qui veut que l'ontologie permet de créer des systèmes de raisonnement sémantique au lieu de syntaxique ?

Nous répondons à la première question en affirmant que l'utilisation de SPARQL pour questionner le système expert comme présenté dans notre méthode entrave la facette intelligente qu'un raisonneur de logique descriptive peut apporter à ce système. Il faut être prudent dans notre utilisation de l'ontologie pour qu'elle ne puisse pas être simplement remplacée par un graphe orienté, mais qu'on utilise plutôt les capacités de raisonnement qu'elle pourvoit. Nous répondons ensuite à la deuxième question en affirmant que le cadre de développement que nous proposons est approprié pour l'interconnexion des systèmes logiques et des modèles d'apprentissage machine. En effet, ce cadre implique des traductions de niveaux d'abstraction effectuées à même l'ontologie et ces traductions peuvent être effectuées par tout algorithme pertinent, y compris par des modèles résultant de l'apprentissage machine. Finalement, nous répondons à la troisième question de manière similaire à la première question. En effet, il est autant possible d'utiliser l'ontologie comme un simple graphe, ce qui n'exploite pas l'aspect sémantique de cette technologie, que comme un système logique déclaratif qui l'exploite.

6.1 Limitations

Une limitation importante de notre travail concerne notre évaluation de la méthode qu'il a produite. En effet, une évaluation plus rigoureuse aurait consisté à comparer ATOM à d'autres méthodes de développement d'ontologie sur des projets semblables avec des équipes de compétences semblables. Cependant, en raison de contraintes de temps, nous n'avons pas pu faire ce genre de test. En effet, étant donné la nature itérative du développement d'une méthode, nous n'avons pu qualifier la méthode d'utilisable que très tard dans le processus et cela ne laissait donc pas assez de temps pour son utilisation plus rigoureuse. De plus, bien qu'elle ait été développée sur deux domaines d'expertise, soit la détection d'intrusion en environnements informatiques et le contrôle aérien, nous n'avons pas pu l'utiliser complètement dans le cas du contrôle aérien. Cela a pour conséquence probable que la méthode est spécialisée pour la modélisation de la connaissance des systèmes informatiques.

Ensuite, nous avons essayé d'évaluer ATOM en fournissant des exigences auxquelles elle doit répondre. Cependant, l'évaluation de ces exigences reste qualitative et biaisée par l'expérience de l'évaluateur. Une évaluation plus rigoureuse de ces exigences par un ensemble d'individus serait nécessaire pour tirer des conclusions plus solides. En effet, même en ce qui concerne l'exigence de temps raisonnable qui se base sur des données quantifiables, le seuil de ce qui est raisonnable ou pas est qualitatif et dépend de dynamiques qui nous échappent. On aurait pu utiliser une étude comparative mais cela n'a pas été fait.

Une autre limitation est que, bien que nous affirmions que les artefacts produits par ATOM permettent l'implémentation d'un système expert, cela n'a pas été formellement testé. Cette conclusion s'appuie plus sur nos propres expériences de développement logiciel et sur ce que nous savons être obligatoire pour l'implémentation de tels systèmes. D'ailleurs, nous pouvons affirmer avoir volontairement laissé un flou sur certaines problématiques. Par exemple, nous ne traitons pas réellement la notion de séquentialité dans les requêtes. Nous avons laissé cette notion à la liberté du développeur puisque cette information est implicite aux séquences de questions.

Une dernière limitation concerne notre constatation qu'il existe tout un domaine de la connaissance au sujet des logiciels centrés sur l'ontologie ("Ontology Driven Software"). Cependant, nous avons fait cette constatation assez tard dans le processus et nous n'avons donc pas pu explorer ce domaine, bien que ce soit exactement ce que nous proposons comme contexte d'utilisation sous la forme d'un système expert. Cela a pour conséquence que ce que nous avançons au sujet du système expert ne se base que sur notre propre expérience et sur l'intuition des experts du domaine de la sécurité informatique impliqués dans notre processus de recherche et non pas sur des résultats de recherches académiques antérieures.

6.2 Améliorations et pistes de recherche futures

Pour répondre à quelques limitations présentées dans la section 6.1, il serait nécessaire d'appuyer nos résultats sur ceux d'une étude comparative qui impliquerait l'utilisation de la méthode proposée dans ce travail par diverses équipes de travail sur divers domaines, afin de limiter les biais d'évaluation. De plus, cela permettrait de répondre aux questions de recherche inter-domaines auxquelles nous n'avons pas pu répondre en raison de notre travail partiel sur le domaine du contrôle aérien.

Il n'est pas prouvé dans ce travail que l'utilisation de l'ontologie en sécurité informatique soit préférable à d'autres solutions. Cependant, nous présentons comme une étape de cette question de recherche une méthode pour développer de manière efficace des ontologies dans

ce contexte. Ce travail devrait donc s'incorporer dans une démarche de recherche plus large ayant pour but de prouver les avantages de l'utilisation de l'ontologie en sécurité informatique et comment l'utiliser pour répondre aux défis actuels de ce domaine.

Une constatation importante que nous avons pu faire durant notre travail concerne l'état quasi artistique du domaine de la représentation de la connaissance. Lorsqu'on arrive concrètement dans le travail de modélisation, on est confronté à un grand nombre de choix pour lesquels il n'y a pas de réponse évidente et justifiable. Cela a pour conséquence qu'il est difficile d'implémenter toute forme de réutilisation ou de recherche de motifs et qu'on fait souvent les choix de modélisation de manière arbitraire. Bref, en général nous comprenons mal les efforts impliqués dans la modélisation et l'utilisation de l'information puisqu'il manque un cadre théorique formel permettant de caractériser cette information. Nous avons tenté de répondre à ce problème en proposant une droite d'abstraction permettant de caractériser l'information et nous avons référé aux langages naturels pour la justification des modélisations non triviales. Il serait cependant intéressant d'étudier si un cadre théorique formel existe pour la représentation de l'information et son utilisation et si le cadre proposé dans notre travail est généralisable et pertinent pour le domaine général de la représentation de l'information. Cela pourrait nous permettre, par exemple, d'évaluer si un système de représentation est plus approprié pour des besoins communicationnels spécifiques qu'un autre.

Finalement, nous avons avancé dans la section 5.2.2 qu'il serait possible d'interconnecter l'ontologie avec des modèles d'apprentissage machine en utilisant notre méthode et dans la section 5.3, qu'elle implique une maintenance facilitée. Ces deux propositions n'ont cependant pas été éprouvées et elles sont énoncées plus à titre d' hypothèses que d'affirmations factuelles. La preuve de ces hypothèses est laissée à une autre étude.

RÉFÉRENCES

F. Abdoli et M. Kahani, “Ontology-based distributed intrusion detection system”, dans *Computer Conference, 2009. CSICC 2009. 14th International CSI.* IEEE, 2009, pp. 65–70.

S. S. Ahmed, “Intrusion alert analysis framework using semantic correlation”, Thèse de doctorat, 2014.

J. An Wang, M. M. Guo, et J. Camargo, “An ontological approach to computer system security”, *Information Security Journal : A Global Perspective*, vol. 19, no. 2, pp. 61–73, 2010.

K. Arbanas et M. Čubrilo, “Ontology in information security”, *Journal of Information and Organizational Sciences*, vol. 39, no. 2, pp. 107–136, 2015.

B. Aristotle *et al.*, *Organon.* Harvard University Press, 1960.

A. Azni, M. M. Saudi, A. Azman, E. M. Tamil, et M. Y. I. Idris, “An efficient network security system through an ontology approach”, dans *Innovations in Information Technology, 2008. IIT 2008. International Conference on.* IEEE, 2008, pp. 267–271.

T. Berners-Lee. (2009) The next web. En ligne : `https://www.ted.com/talks/tim_berners_lee_on_the_next_web`

C. Biemann, “Ontology learning from text : A survey of methods.” dans *LDV forum*, vol. 20, no. 2, 2005, pp. 75–93.

G. Booch, *Object oriented analysis & design with application.* Pearson Education India, 2006.

P. Bourque, R. E. Fairley, A. Abran, J. Garbajosa, G. Keeni, B. Shen, et A. April, “Guide to the software engineering body of knowledge”, 2014.

M. Busch et M. Wirsing, “An ontology for secure web applications.” *Int. J. Software and Informatics*, vol. 9, no. 2, pp. 233–258, 2015.

R. Carnap, “Empirisme, sémantique et ontologie”, *Signification et nécessité*, pp. 311–335, 1997.

E. W. Dijkstra *et al.*, “On the cruelty of really teaching computing science”, *Communications of the ACM*, vol. 32, no. 12, pp. 1398–1404, 1989.

É. Ducharme, “Détection d’intrusion a l’aide d’un systeme expert basé sur une ontologie”, Thèse de doctorat, École Polytechnique de Montréal, 2017.

G. Elahi, E. Yu, et N. Zannone, “A modeling ontology for integrating vulnerabilities into security requirements conceptual foundations”, dans *International Conference on Conceptual Modeling*. Springer, 2009, pp. 99–114.

S. Fenz et A. Ekelhart, “Formalizing information security knowledge”, dans *Proceedings of the 4th international Symposium on information, Computer, and Communications Security*. ACM, 2009, pp. 183–194.

G. Fenza, D. Furno, V. Loia, et M. Veniero, “Agent-based cognitive approach to airport security situation awareness”, dans *Complex, Intelligent and Software Intensive Systems (CISIS), 2010 International Conference on*. IEEE, 2010, pp. 1057–1062.

M. Fernández-López, A. Gómez-Pérez, et N. Juristo, “Methontology : from ontological art towards ontological engineering”, 1997.

J.-b. Gao, B.-w. Zhang, X.-h. Chen, et Z. Luo, “Ontology-based model of network and computer attacks for security assessment”, *Journal of Shanghai Jiaotong University (Science)*, vol. 18, no. 5, pp. 554–562, 2013.

P. Garcia-Teodoro, J. Diaz-Verdejo, G. Maciá-Fernández, et E. Vázquez, “Anomaly-based network intrusion detection : Techniques, systems and challenges”, *computers & security*, vol. 28, no. 1, pp. 18–28, 2009.

N. Goodman, “Seven strictures on similarity”, 1972.

T. R. Gruber, “A translation approach to portable ontology specifications”, *Knowledge acquisition*, vol. 5, no. 2, pp. 199–220, 1993.

Y. Hollander. (2017, July) Where machine learning meets rule-based verification.

G. A. Isaza, A. G. Castillo, M. López, et L. F. Castillo, “Towards ontology-based intelligent model for intrusion detection and prevention.” dans *CISIS*. Springer, 2009, pp. 109–116.

D. Jones, T. Bench-Capon, et P. Visser, “Methodologies for ontology development”, 1998.

D. R. Karger, “The semantic web and end users : What’s wrong and how to fix it”, *IEEE Internet Computing*, vol. 18, no. 6, pp. 64–70, 2014.

M. Klauza, P. Czekalski, et K. Tokarz, “Air traffic data integration using the semantic web approach”, *Athens Journal of Technology & Engineering*, 2014.

H. J. Levesque, “Knowledge representation and reasoning”, dans *Readings in Artificial Intelligence and Databases*. Elsevier, 1988, pp. 35–51.

F.-H. Liu, “Constructing enterprise information network security risk management mechanism by using ontology”, dans *Advanced Information Networking and Applications Workshops, 2007, AINAW’07. 21st International Conference on*, vol. 1. IEEE, 2007, pp. 929–934.

R. Luh, S. Marschalek, M. Kaiser, H. Janicke, et S. Schrittwieser, “Semantics-aware detection of targeted attacks : a survey”, *Journal of Computer Virology and Hacking Techniques*, vol. 13, no. 1, pp. 47–85, 2017.

A. Martimiano et E. Moreira, “An owl-based security incident ontology”, dans *Proceedings of the Eighth International Protege Conference*, 2005, pp. 43–44.

F. Meyer, R. Kroeger, R. Heidger, et M. Milekovic, “An approach for knowledge-based it management of air traffic control systems”, dans *Network and Service Management (CNSM), 2013 9th International Conference on*. IEEE, 2013, pp. 345–349.

A. Monnin, “L’ingénierie philosophique de rudolf carnap”, *Cahiers philosophiques*, no. 2, pp. 27–53, 2015.

T. Moser, R. Mordinyi, A. Mikula, et S. Biffl, “Making expert knowledge explicit to facilitate tool support for integrating complex information systems in the atm domain”, dans *Complex, Intelligent and Software Intensive Systems, 2009. CISIS’09. International Conference on*. IEEE, 2009, pp. 90–97.

N. F. Noy, D. L. McGuinness *et al.*, “Ontology development 101 : A guide to creating your first ontology”, 2001.

T. S. M. Pereira et H. M. D. Santos, “An ontology approach in designing security information systems to support organizational security risk knowledge.” dans *KEOD*, 2012, pp. 461–466.

M. Poveda-Villalón, M. Suárez-Figueroa, et A. Gómez-Pérez, “Validating ontologies with oops !” *Knowledge Engineering and Knowledge Management*, pp. 267–281, 2012.

W. V. O. Quine, *From a Logical Point of View : 9 [nine] Logico-philosophical Essays.* Mass., 1971.

Y. Raimond et G. Schreiber, "RDF 1.1 primer", W3C, W3C Note, jun 2014, http ://www.w3.org/TR/2014/NOTE-rdf11-primer-20140624/.

V. Raskin, C. F. Hempelmann, K. E. Triezenberg, et S. Nirenburg, "Ontology in information security : a useful theoretical foundation and methodological tool", dans *Proceedings of the 2001 workshop on New security paradigms.* ACM, 2001, pp. 53–59.

B. Russell, "On denoting", *Mind*, vol. 14, no. 56, pp. 479–493, 1905.

S. Russell et P. Norvig, *Intelligence artificielle : Avec plus de 500 exercices.* Pearson Education France, 2010.

S. Saad et I. Traore, "Method ontology for intelligent network forensics analysis", dans *Privacy Security and Trust (PST), 2010 Eighth Annual International Conference on.* IEEE, 2010, pp. 7–14.

A. Sadighian, "Intrusion detection from heterogenous sensors", Thèse de doctorat, École Polytechnique de Montréal, 2015.

A. Simmonds, P. Sandilands, et L. Van Ekert, "An ontology for network security attacks", dans *Asian Applied Computing Conference.* Springer, 2004, pp. 317–323.

C. Simmons, C. Ellis, S. Shiva, D. Dasgupta, et Q. Wu, "Avoidit : A cyber attack taxonomy", 2009.

C. B. Simmons, S. G. Shiva, et L. L. Simmons, "A qualitative analysis of an ontology based issue resolution system for cyber attack management", dans *Cyber Technology in Automation, Control, and Intelligent Systems (CYBER), 2014 IEEE 4th Annual International Conference on.* IEEE, 2014, pp. 323–329.

M. Stefik, *Introduction to knowledge systems.* Morgan Kaufman, 1995.

K.-W. Su, H.-Y. Wang, K.-J. Li, C.-H. Wang, et P.-H. Hsiao, "Designing and evaluating an ontology-based air traffic control digital knowledge learning system", dans *User Science and Engineering (i-USEr), 2011 International Conference on.* IEEE, 2011, pp. 172–177.

L. Torvalds. (2006) Re : Licensing and the library version of git. En ligne : `https://lwn.net/Articles/193245/`

M. Uschold et M. Gruninger, “Ontologies : Principles, methods and applications”, *The knowledge engineering review*, vol. 11, no. 2, pp. 93–136, 1996.

J. Van Eijck et C. Unger, *Computational semantics with functional programming*. Cambridge University Press, 2010.

R. P. van Heerden, B. Irwin, et I. Burke, “Classifying network attack scenarios using an ontology”, dans *Proceedings of the 7th International Conference on Information-Warfare & Security (ICIW 2012)*, 2012, pp. 311–324.

W3C, “Sparql 1.1 overview”, W3C, W3C Recommendation, Mars 2013, http ://www.w3.org/TR/2013/REC-sparql11-overview-20130321/.

H. Xu, D. Xiao, et Z. Wu, “Application of security ontology to context-aware alert analysis”, dans *Computer and Information Science, 2009. ICIS 2009. Eighth IEEE/ACIS International Conference on*. IEEE, 2009, pp. 171–176.